

# Java

## 典型模块与项目实战大全

明日科技 李伟 王国辉 编著

电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间：可以联系群主获取更多大型企业内部技术教程。

## 内 容 简 介

本书以 Java 主流技术应用及项目开发为主线,通过 Java 开发中最常见的 15 个典型模块和 4 个完整的项目案例,详细介绍了常用模块和项目开发的完整过程。全书分为典型模块篇、项目实战篇和环境搭建篇等 3 篇,共 24 章。典型模块篇包括备忘录模块、学生成绩管理模块、常用照片管理模块、定制打印模块、短信收发模块、FTP 上传下载模块、局域网通信模块、区域地图模块、序列号注册模块、PDF 查看模块、动态考题模块、多功能查询模块、文件分割模块、图书管理模块、五子棋游戏模块等 15 个模块,每个模块都分步进行了详细介绍,关键技术还进行了重点讲解。项目实战篇包括酒店管理系统、企业人事管理系统、医药综合管理系统、进销存管理系统等 4 个完整项目,每个项目都从软件工程的角度出发,从开发背景、需求分析、系统功能分析、数据库分析、数据库结构、系统开发到系统的编译发行,每一过程都进行了详细介绍。环境搭建篇包括 JDK 下载与安装、Eclipse 下载与汉化、Eclipse 配置与插件安装、MySQL 及其工具下载安装、SQL Server 2005 数据库的安装,主要对开发环境的搭建及程序开发前的必备基础知识进行了讲解。

本书所附配套光盘提供了书中所有案例的全部源代码,所有源代码都经过精心调试,在 Windows XP、Windows 7 和 Windows 2003 下全部运行通过,保证能够正常运行。

本书适合 Java 程序员参考学习,也可作为高等院校相关专业师生的参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

### 图书在版编目(CIP)数据

Java 典型模块与项目实战大全 / 明日科技等编著. —北京: 电子工业出版社, 2012.4  
ISBN 978-7-121-16112-4

I. ①J… II. ①明… III. ①Java 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2012) 第 033331 号

策划编辑: 胡辛征

责任编辑: 许 艳

特约编辑: 赵树刚

印 刷:

装 订: 北京中新伟业印刷有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 35 字数: 903 千字

印 次: 2012 年 4 月第 1 次印刷

印 数: 4000 册 定价: 79.00 元 (含 DVD 光盘 1 张)

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线: (010) 88258888。



# 前言

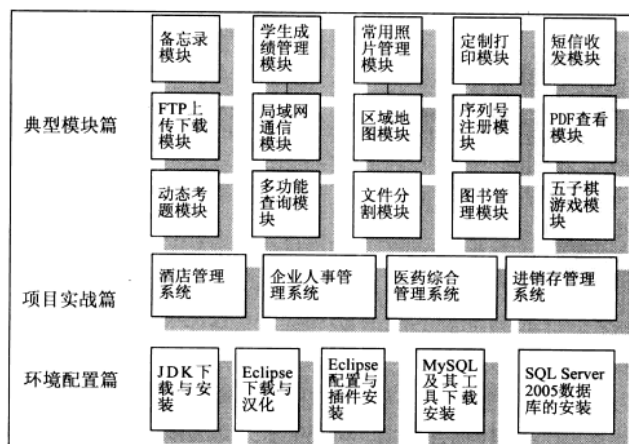
当前，世界已步入信息化时代，作为信息产业核心的软件及信息服务产业，已成为新世纪全球最重要，也是最核心的部分。无论是进行宇宙探索的航天飞机，还是围绕在我们身边的各种汽车、家电或手机，都已越来越离不开程序和代码。同时，软件开发工作是相当复杂的，一个软件项目，其开发过程是极其艰难的，不但充斥着需要随时解决的各种技术难题，更会面临无时不在的各种风险压力。一个有经验的开发人员，除了应用自身的积累和经验，还需经常借鉴和参考其他软件项目的开发经验。

目前，我国软件产业蓬勃发展，越来越多的人进入到软件开发领域，他们迫切需要了解各类项目的开发过程和经验。而国内介绍相关项目开发和学习的图书相对较少，为了使读者摆脱这方面的窘境，我们编写了这本书。本书作者精选了 15 个实用模块和 4 个项目，涵盖了各种应用环境，是进行项目开发必不可少的参考书。“宝剑赠壮士”，希望本书提供的各类项目或模块能给在开发征途中自强不息、奋斗不止的学习者一点点借鉴、一点点启发。学习的目的是为了实践，也是为了应用到实际工作中。希望本书能鞭策学习者，在牢固掌握基本开发技能的同时，提高实战开发能力——“磨刀不误砍柴工”。



## 「本书内容」

本书分为 3 篇 24 章内容，包括 15 个典型模块，4 个完整项目和 5 章环境搭建知识，具体如下图所示。





## 【本书特色】

### ☑ 模块丰富，实用超值

本书精选了 15 个典型模块，涵盖了多个方面的应用，这些模块稍加修改便可使用，也可以作为某个项目的重要组成部分。

### ☑ 项目完整，提高实战水平

本书提供了 4 个完整项目，进一步提高读者的实战水平。每个项目都从开发者角度出发，讲解完整，贴近实际。

### ☑ 易学、易用

书中提供了关键代码解析，对代码中重要的对象、方法、语句和重点知识等进行说明，以便于读者在阅读代码时透彻理解代码的含义和相关技术、技巧。

### ☑ 赠送所有模块和项目源代码

书中所有模块和项目均提供源代码，用户在开发中可以快速借鉴或应用。

### ☑ 段落工整、简洁，图文结合，更容易学习

### ☑ 编码规范，注释详尽

为了提高用户的实际开发能力，书中代码都是经过严格审查的，可以与商业源代码媲美，并且为了方便用户阅读代码，几乎所有代码都提供详细注释。



## 【超值 DVD 光盘】

为了帮助读者学习和使用书中提供的模块和项目，本书附赠有 DVD 光盘，里面提供书中所有典型模块和项目的源代码。光盘目录如下图所示。



## 【本书约定】

### ☑ 模块和项目使用方法

用户在学习本书过程中，可以从光盘中复制模块和项目，去掉其只读属性。有些模块或项目需要使用相应的数据库或第三方资源，这些模块或项目在使用前需要进行相应

配置，详细使用方式请参考《Java 典型模块与项目实战大全》光盘使用说明。此外，如果用户直接将本书模块或项目用于商业用途，由此产生的不良后果由用户自己承担。

☑ 部分模块或项目只给出关键代码

由于篇幅限制，书中有些模块或项目只给出了关键代码，完整代码请参考光盘中的源程序。



## 「本书适用人群」

本书非常适合以下人员阅读：

- ☑ 从事 Java 编程工作的开发人员
- ☑ 有一定语言基础，想进一步提高技能的人员
- ☑ 大中专院校的老师和学生
- ☑ 即将走向工作岗位的大学毕业生
- ☑ 相关培训机构的老师和学员
- ☑ Java 编程爱好者



## 「在线互动答疑」

如果您在学习或使用本书的过程中遇到问题或疑惑，可以通过如下方式与我们联系。

- ☑ 服务网站：[www.mingribook.com](http://www.mingribook.com)
- ☑ 服务 QQ：1026560213
- ☑ 服务信箱：[mingrisoft@mingrisoft.com](mailto:mingrisoft@mingrisoft.com)
- ☑ 服务电话：0431-84978981/84978982
- ☑ 学习社区：[www.mrbccd.com](http://www.mrbccd.com)

我们承诺将在 5 个工作日内给您提供解答。



## 「本书作者」

本书由明日科技组织编写，参加编写的人员有李伟、陈丹丹、王国辉、高春艳、王小科、赵会东、聂喜婷、李继业、赛奎春、潘凯华、刘欣、李慧、杨丽、刘龄龄、陈英、朱晓等。由于作者水平有限，疏漏和不足之处在所难免，请广大读者朋友批评指正。

明日科技  
2012 年 2 月

• V •

# 第 18 章

---

## 医药综合管理系统

( Hibernate+Spring+Server 2005 实现 )

随着科技的发展，人们越来越关注自身的健康。医药行业也因此有了跨越性的发展。面对种类繁多的药品，使用人工方式管理其进存销是非常麻烦的。不仅耗时耗力，而且安全性也得不到保障。医药综合系统作为良好的解决方案便应运而生。本章将使用新版的 Hibernate 和 Spring 框架完成这样一个系统。通过本章的学习，读者可以掌握以下内容：

- » 如何进行需求分析和编写项目计划书
- » 进销存管理系统的一般设计理念
- » 如何分析并设计数据库
- » Hibernate 框架的使用
- » Spring 框架的使用
- » Hibernate 与 Spring 框架的整合开发
- » 使用 Swing 技术开发桌面应用程序

内容参见光盘



# 第 19 章

---

## 进销存管理系统

( Swing+SQL Server 2005 实现 )

实现企业信息化管理是现代社会中小企业稳步发展的必要条件，它可以提高企业的管理水平和工作效率，最大限度地减少手工操作带来的失误。进销存管理系统正是一个信息化管理软件，可以实现企业的进货、销售、库存管理等各项业务的信息化管理。本章将介绍如何使用 Java Swing 技术和 SQL Server 2005 数据库开发跨平台的应用程序。通过阅读本章，读者可以学习到：

- » 如何进行项目的可行性分析
- » 如何系统设计
- » 如何进行数据库分析和数据库建模
- » 企业进销存主要功能模块的开发过程
- » 如何设计公共类
- » 如何将程序打包

内容参见光盘

# 第三篇

## 环境搭建篇

本篇主要内容：

- 第 20 章 JDK 下载与安装（内容参见光盘）
- 第 21 章 Eclipse 下载与汉化（内容参见光盘）
- 第 22 章 Eclipse 配置与插件安装（内容参见光盘）
- 第 23 章 MySQL 及其工具下载安装（内容参见光盘）
- 第 24 章 SQL Server 2005 安装（内容参见光盘）

内容参见光盘

书山有路  
勤为径  
学海无涯  
苦作舟

# 目 录

## 第一篇 典型模块篇

第 1 章 备忘录模块 (Swing+JDBC 实现)	2
1.1 备忘录模块概述	3
1.1.1 模块概述	3
1.1.2 功能结构	3
1.1.3 程序预览	3
1.2 关键技术	4
1.2.1 绘制艺术字	4
1.2.2 窗体居中显示	5
1.2.3 使用 JavaBean 封装信息	6
1.2.4 获得 MySQL 数据库连接	7
1.2.5 批量处理数据库操作	7
1.2.6 使用 List 保存查询结果	9
1.2.7 使用正则表达式校验日期	11
1.2.8 调用系统工具	12
1.3 主窗体	12
1.3.1 功能概述	12
1.3.2 添加菜单及菜单项	13
1.3.3 绘制窗体中的艺术字	14
1.3.4 设置窗体显示位置和大小	15
1.4 增加备忘录	15
1.4.1 功能概述	15
1.4.2 设置文本框和文本区控件	15
1.4.3 添加工具按钮	17
1.4.4 保存备忘录信息	17
1.4.5 清空备忘录信息	18
1.4.6 销毁窗体	19
1.5 修改备忘录	19
1.5.1 功能概述	19
1.5.2 设置文本框和文本区控件	19
1.5.3 添加工具按钮	21
1.5.4 填充备忘录信息	21
1.5.5 修改前一条备忘录信息	22
1.5.6 修改后一条备忘录信息	22
1.5.7 修改备忘录信息	23
1.6 查询备忘录	24
1.6.1 功能概述	24
1.6.2 设置文本框和文本区控件	24
1.6.3 添加工具按钮	25
1.6.4 查询备忘录信息	26
1.7 显示查询结果	27
1.7.1 功能概述	27
1.7.2 设置文本框和文本区控件	28
1.7.3 添加工具按钮	29
1.7.4 填充备忘录信息	29
1.7.5 查看上一条查询结果	30
1.7.6 查看下一条查询结果	30
1.8 删除备忘录	31
1.8.1 功能概述	31
1.8.2 设置文本框和文本区控件	31
1.8.3 添加工具按钮	32
1.8.4 填充备忘录信息	33
1.8.5 删除上一条备忘录信息	33
1.8.6 删除下一条备忘录信息	33
1.8.7 删除备忘录信息	34
第 2 章 学生成绩管理模块 (Swing+MySQL 实现)	35
2.1 学生成绩管理模块概述	36
2.1.1 模块概述	36
2.1.2 功能结构	36
2.1.3 程序预览	36

2.2	关键技术.....	38	2.4.5	清空成绩单信息.....	51
2.2.1	绘制艺术字.....	38	2.4.6	销毁窗体.....	52
2.2.2	窗体居中显示.....	38	2.5	显示已保存成绩单.....	52
2.2.3	使用 JavaBean 封装信息.....	39	2.2.1	功能概述.....	52
2.2.4	获得 MySQL 数据库连接.....	40	2.2.2	为表格控件添加数据.....	53
2.2.5	批量处理数据库操作.....	41	2.2.3	“修改”按钮事件监听.....	53
2.2.6	使用 List 保存查询结果.....	42	2.2.4	“删除”按钮事件监听.....	54
2.2.7	使用正则表达式进行校验.....	44	2.6	修改成绩单.....	55
2.2.8	调用系统工具.....	45	2.6.1	功能概述.....	55
2.3	主窗体.....	46	2.6.2	设置文本框控件.....	55
2.3.1	功能概述.....	46	2.6.3	添加工具按钮.....	57
2.3.2	添加菜单及菜单项.....	46	2.6.4	填充成绩单信息.....	57
2.3.3	绘制窗体中的艺术字.....	47	2.6.5	修改成绩单信息.....	57
2.3.4	设置窗体显示位置和大小.....	48	2.7	查询成绩单.....	59
2.4	增加成绩单.....	48	2.7.1	功能概述.....	59
2.4.1	功能概述.....	48	2.7.2	设置文本框控件.....	59
2.4.2	配置非按钮控件.....	48	2.7.3	添加工具按钮.....	61
2.4.3	添加工具按钮.....	50	2.7.4	查询成绩单信息.....	61
2.4.4	保存成绩单信息.....	50	2.7.5	显示查询结果.....	62
第3章	常用照片管理模块（Swing+图片处理技术实现）.....	64	3.3.3	修改相册.....	77
3.1	常用照片管理模块概述.....	65	3.3.4	删除相册.....	78
3.1.1	设计思路.....	65	3.4	照片管理.....	78
3.1.2	功能结构.....	65	3.4.1	功能概述.....	78
3.1.3	效果预览.....	66	3.4.2	添加照片.....	79
3.2	关键技术.....	68	3.4.3	修改照片信息.....	80
3.2.1	捕获树的选中节点事件.....	68	3.4.4	删除照片.....	81
3.2.2	捕获树的展开节点事件.....	69	3.4.5	搜索照片.....	82
3.2.3	浏览方式切换技术.....	70	3.4.6	保存照片.....	85
3.2.4	随意选取照片技术.....	72	3.5	照片显示.....	86
3.2.5	照片缩放与内存溢出.....	75	3.5.1	功能概述.....	86
3.2.6	换行显示提示信息.....	76	3.5.2	全屏显示照片.....	86
3.3	相册树.....	76	3.5.3	照片播放器.....	87
3.3.1	功能概述.....	76			
3.3.2	添加相册.....	77			
第4章	定制打印模块（Swing+MySQL 实现）.....	90			
4.1	定制打印模块概述.....	91	4.2.2	窗体居中显示.....	94
4.1.1	模块概述.....	91	4.2.3	使用 JavaBean 封装信息.....	94
4.1.2	功能结构.....	91	4.2.4	获得 MySQL 数据库连接.....	96
4.1.3	程序预览.....	91	4.2.5	批量处理数据库操作.....	96
4.2	关键技术.....	93	4.2.6	使用 List 保存查询结果.....	98
4.2.1	自定义面板背景图片.....	93	4.2.7	使用 Java 操作打印机.....	99
			4.3	主窗体.....	100



4.3.1	功能概述 .....	100	4.5.4	添加工具按钮 .....	109
4.3.2	添加菜单及菜单项 .....	100	4.5.5	填充快递单信息 .....	109
4.3.3	加载窗体背景图片 .....	101	4.5.6	获得上一条快递单信息 .....	110
4.3.4	设置窗体显示位置和大小 .....	102	4.5.7	获得下一条快递单信息 .....	110
4.4	添加快递单 .....	102	4.5.8	修改快递单信息 .....	111
4.4.1	功能概述 .....	102	4.6	打印快递单 .....	112
4.4.2	加载快递单图片 .....	103	4.6.1	功能概述 .....	112
4.4.3	设置文本框和文本域控件 .....	103	4.6.2	加载快递单图片 .....	113
4.4.4	添加工具按钮 .....	104	4.6.3	设置文本框和文本域控件 .....	113
4.4.5	保存快递单信息 .....	104	4.6.4	添加工具按钮 .....	114
4.4.6	清空快递单信息 .....	106	4.6.5	填充快递单信息 .....	114
4.4.7	销毁窗体 .....	106	4.6.6	获得上一条快递单信息 .....	115
4.5	修改快递单 .....	107	4.6.7	获得下一条快递单信息 .....	115
4.5.1	功能概述 .....	107	4.6.8	分割文本区信息 .....	116
4.5.2	加载快递单图片 .....	107	4.6.9	打印快递单信息 .....	116
4.5.3	设置文本框和文本域控件 .....	108			
第 5 章	短信收发模块 (Swing+GSM MODEM 实现) .....	119			
5.1	短信收发模块概述 .....	120	5.4.3	显示短信 .....	133
5.1.1	模块概述 .....	120	5.5	发送短信 .....	134
5.1.2	功能结构 .....	120	5.5.1	功能概述 .....	134
5.1.3	程序预览 .....	120	5.5.2	添加删除收信人 .....	134
5.2	关键技术 .....	121	5.5.3	“联系人”选项卡 .....	135
5.2.1	短信猫技术 .....	121	5.5.4	编写短信内容 .....	136
5.2.2	收发短信 .....	123	5.6	发信箱 .....	137
5.2.3	选项卡的关联 .....	126	5.6.1	功能概述 .....	137
5.2.4	卡片布局 .....	127	5.6.2	读取已发短信 .....	137
5.2.5	树控件的使用 .....	129	5.6.3	显示收信人列表 .....	139
5.3	设置并连接短信猫 .....	129	5.7	联系人管理 .....	139
5.3.1	功能概述 .....	129	5.7.1	功能概述 .....	139
5.3.2	短信猫设置 .....	130	5.7.2	添加联系人组别 .....	140
5.3.3	连接短信猫 .....	131	5.7.3	联系人对话框 .....	141
5.4	读取短信 .....	132	5.7.4	添加联系人 .....	145
5.4.1	功能概述 .....	132	5.7.5	修改联系人或组别 .....	145
5.4.2	读取短信 .....	132	5.7.6	删除联系人或组别 .....	145
第 6 章	FTP 上传下载模块 (Swing+FTP 技术实现) .....	147			
6.1	FTP 上传下载模块概述 .....	148	6.2.3	浏览服务器资源 .....	152
6.1.1	模块概述 .....	148	6.2.4	FTP 文件上传与下载 .....	153
6.1.2	功能结构 .....	148	6.2.5	向 FTP 服务器发送命令 .....	155
6.1.3	系统预览 .....	149	6.2.6	获取文件在本系统的 显示图标 .....	157
6.2	关键技术 .....	150	6.2.7	任务队列 .....	158
6.2.1	登录 FTP 服务器 .....	150	6.3	FTP 站点管理 .....	160
6.2.2	浏览本地资源 .....	151			

6.3.1 功能概述	160	6.5.2 删除服务器文件	170
6.3.2 读取属性文件	161	6.5.3 重命名服务器文件或文件夹	171
6.3.3 装载FTP站点信息	161	6.5.4 新建文件夹	172
6.3.4 添加FTP站点	162	6.5.5 添加服务器资源到下载队列	172
6.4 本地资源管理	163	6.5.6 刷新服务器资源列表	173
6.4.1 功能概述	163	6.6 队列管理	174
6.4.2 删除本地文件	164	6.6.1 功能概述	174
6.4.3 重命名本地文件或文件夹	165	6.6.2 任务队列	175
6.4.4 新建文件夹	166	6.6.3 本地队列文件上传	178
6.4.5 添加本地文件到上传队列	167	6.6.4 FTP队列文件下载	181
6.4.6 刷新本地资源列表	168		
6.5 FTP资源管理	169		
6.5.1 功能概述	169		
第7章 局域网通信模块 (Swing+Java DB 实现)	184		
7.1 局域网通信概述	185	7.5 实现系统工具	199
7.1.1 模块概述	185	7.5.1 功能概述	199
7.1.2 功能结构	185	7.5.2 实现界面选择	200
7.1.3 程序预览	185	7.5.3 实现搜索新用户	200
7.2 关键技术	186	7.5.4 进行系统操作	201
7.2.1 创建操作数据库的Dao类	186	7.6 用户管理	203
7.2.2 工具类的实现	190	7.6.1 功能概述	203
7.3 主窗体	194	7.6.2 创建用户树列表	204
7.3.1 功能概述	194	7.6.3 在用户树中显示用户	206
7.3.2 实现主窗体	194	7.6.4 从用户树中删除用户	206
7.3.3 记录窗体位置	197	7.6.5 向用户树中添加用户	207
7.4 实现系统托盘	197	7.7 实现通信	208
7.4.1 功能概述	197	7.7.1 功能概述	208
7.4.2 初始化系统托盘	198	7.7.2 实现通信窗体	208
7.4.3 实现弹出菜单	198	7.7.3 接收信息	210
7.4.4 双击托盘图标显示主窗体	199	7.7.4 发送信息	210
		7.7.5 系统信使	211
第8章 区域地图模块 (Swing+Java DB+绘图技术实现)	212		
8.1 区域地图模块概述	213	8.2.5 维护树模型	224
8.1.1 设计思路	213	8.3 地图处理器	226
8.1.2 功能结构	213	8.3.1 功能概述	226
8.1.3 程序预览	214	8.3.2 获得小地图	226
8.2 关键技术	216	8.3.3 处理缩放和显示位置	228
8.2.1 Java DB数据库	216	8.4 地图显示	229
8.2.2 万年历选择框技术	217	8.4.1 功能概述	229
8.2.3 滑块控件使用技术	222	8.4.2 绘制大地图	230
8.2.4 列表控件使用	223	8.4.3 绘制小地图	232

8.5 地图操作 .....	233	8.6.4 删除标记 .....	240
8.5.1 功能概述 .....	233	8.6.5 查看标记信息 .....	240
8.5.2 实现地图缩放功能 .....	233	8.7 标记搜索 .....	241
8.5.3 实现地图移动功能 .....	235	8.7.1 功能概述 .....	241
8.6 标记维护 .....	236	8.7.2 常用搜索 .....	242
8.6.1 功能概述 .....	236	8.7.3 高级搜索 .....	243
8.6.2 创建弹出菜单 .....	236	8.7.4 描红并居中显示标记 .....	244
8.6.3 创建和修改标记 .....	238		
第9章 序列号注册模块 (Swing+RSA 实现) .....	246		
9.1 序列号注册模块概述 .....	247	9.2.8 RSA 加密/解密算法 .....	255
9.1.1 模块概述 .....	247	9.3 软件注册导航窗体 .....	257
9.1.2 功能结构 .....	247	9.3.1 功能概述 .....	257
9.1.3 程序预览 .....	247	9.3.2 软件试用功能实现 .....	259
9.2 关键技术 .....	248	9.4 软件注册窗体 .....	261
9.2.1 读取客户端 MAC 地址 .....	248	9.4.1 功能概述 .....	261
9.2.2 Java 操作注册表 .....	249	9.4.2 验证注册码 .....	264
9.2.3 避免用户修改系统时间 .....	250	9.4.3 限制使用时间 .....	266
9.2.4 弹出菜单 .....	251	9.4.4 保证使用唯一性 .....	267
9.2.5 一次性粘贴注册码 .....	253	9.5 注册机实现 .....	268
9.2.6 计算两个时间的 间隔天数 .....	253	9.5.1 功能概述 .....	268
9.2.7 ini 文件的读写 .....	254	9.5.2 生成注册码 .....	269
第10章 PDF 查看模块 (Swing+PDF Render 实现) .....	271		
10.1 PDF 查看模块概述 .....	272	10.4 打开 PDF 文档 .....	288
10.1.1 模块概述 .....	272	10.4.1 功能概述 .....	288
10.1.2 功能结构 .....	272	10.4.2 创建文件选择器 .....	288
10.1.3 程序预览 .....	272	10.4.3 在文件选择器中只显示 PDF 文档 .....	290
10.2 关键技术 .....	274	10.4.4 使窗体标题栏显示 PDF 文档名称 .....	290
10.2.1 PDF Render 组件技术 .....	274	10.4.5 显示 PDF 文档内容 .....	291
10.2.2 实现 PDF 文档缩放 .....	274	10.5 缩位图导航 .....	291
10.2.3 实现 PDF 文档分页 .....	277	10.5.1 功能概述 .....	291
10.2.4 实现 PDF 文档打印、 页面设置 .....	280	10.5.2 实现缩位图面板 .....	292
10.2.5 实现 PDF 文档自动 滚动功能 .....	282	10.5.3 实现缩位图索引功能 .....	292
10.2.6 实现抓手功能 .....	283	10.6 书签导航 .....	293
10.3 主窗体 .....	285	10.6.1 功能概述 .....	293
10.3.1 功能概述 .....	285	10.6.2 实现书签面板 .....	294
10.3.2 菜单栏的实现 .....	285	10.6.3 实现书签索引功能 .....	294
10.3.3 工具栏的实现 .....	286	10.7 全屏显示 PDF 文档 .....	295
10.3.4 左侧索引面板的实现 .....	287	10.7.1 功能概述 .....	295
10.3.5 右侧界面的实现 .....	287		



10.7.2 在工具栏中添加 “全屏”按钮.....	296	10.7.3 实现全屏显示功能 .....	296
<b>第 11 章 动态考题模块 (Swing+MySQL 实现) .....</b>	<b>300</b>		
11.1 动态考题模块概述 .....	301	11.4.5 实现自动阅卷 .....	320
11.1.1 模块概述 .....	301	11.5 管理员查分功能.....	322
11.1.2 功能结构 .....	301	11.5.1 功能概述 .....	322
11.1.3 程序预览 .....	301	11.5.2 按考生号查询成绩 .....	322
11.2 关键技术 .....	304	11.5.3 按考生姓名查询考分 .....	323
11.2.1 设置窗体背景 .....	304	11.6 添加考题.....	325
11.2.2 编写字符处理类 .....	304	11.6.1 功能概述 .....	325
11.2.3 编写获取时间方法 .....	305	11.6.2 使用 List 集合存储 所有考题 .....	325
11.2.4 编写 Java Bean.....	306	11.6.3 自动计算考题号 .....	326
11.2.5 倒计时 .....	307	11.6.4 保存考题 .....	326
11.3 登录窗体 .....	308	11.7 修改/删除考题设计.....	328
11.3.1 功能概述 .....	308	11.7.1 功能概述 .....	328
11.3.2 编写验证用户是否合法的 方法 .....	309	11.7.2 实现修改考题 .....	328
11.3.3 实现登录功能 .....	310	11.7.3 实现删除试题 .....	330
11.4 考试主窗体 .....	312	11.8 考试参数设置.....	331
11.4.1 功能概述 .....	312	11.8.1 功能概述 .....	331
11.4.2 显示考生姓名 .....	313	11.8.2 在下拉列表中显示内容 .....	331
11.4.3 显示考题 .....	314	11.8.3 实现考试参数设置 .....	333
11.4.4 转到上一题、下一题 .....	317		
<b>第 12 章 多功能查询模块 (Swing+SQL Server 2005 实现) .....</b>	<b>335</b>		
12.1 多功能查询模块概述 .....	336	12.3 文件操作.....	343
12.1.1 模块概述 .....	336	12.3.1 功能概述 .....	343
12.1.2 业务流程 .....	336	12.3.2 实现向 txt 文件中写数据 .....	343
12.1.3 程序预览 .....	337	12.3.3 实现将查询结果写入 txt 文件中 .....	344
12.2 关键技术 .....	337	12.4 事件处理.....	347
12.2.1 JDBC 技术 .....	337	12.4.1 功能概述 .....	347
12.2.2 查询语句结构 .....	339	12.4.2 实现获取表中的字段 描述信息 .....	347
12.2.3 获取字段的描述信息 .....	340	12.4.3 实现绑定组件的 处理事件 .....	348
12.2.4 获取数据库中的 所有表名 .....	340	12.4.4 显示调用程序窗体 .....	349
12.2.5 如何将程序加载到 其他程序中 .....	341		
<b>第 13 章 文件分割模块 (Swing+I/O 文件处理技术实现) .....</b>	<b>350</b>		
13.1 文件分割模块概述 .....	351	13.2 关键技术.....	352
13.1.1 模块概述 .....	351	13.2.1 文件操作与读写 .....	352
13.1.2 功能结构 .....	351	13.2.2 获取系统有效盘符 .....	355
13.1.3 程序预览 .....	351	13.2.3 转换文件编码格式 .....	355



13.2.4	文件解压缩 .....	357	13.7.3	实现移动整个文件夹 .....	376
13.2.5	表格控件的使用 .....	359	13.8	实现批量删除 .....	378
13.3	主窗体 .....	361	13.8.1	功能概述 .....	378
13.3.1	功能概述 .....	361	13.8.2	“扩展名”下拉 列表设计 .....	378
13.3.2	菜单栏设计 .....	361	13.8.3	文件日期文本框设计 .....	379
13.3.3	工具栏设计 .....	362	13.9	实现批量重命名 .....	381
13.3.4	实现显示系统文件夹 .....	363	13.9.1	功能概述 .....	381
13.3.5	实现显示系统文件夹中的 文件 .....	365	13.9.2	批量重命名文件 .....	382
13.4	新建文件 .....	366	13.10	批量修改文件编码格式 .....	384
13.4.1	功能概述 .....	366	13.10.1	功能概述 .....	384
13.4.2	实现新建文件 .....	367	13.10.2	批量修改文件编码 .....	384
13.4.3	实现新建文件夹 .....	368	13.11	压缩和解压缩文件 .....	385
13.5	实现文件搜索 .....	369	13.11.1	功能概述 .....	385
13.5.1	功能概述 .....	369	13.11.2	实现压缩文件 .....	385
13.5.2	在下拉列表中添加 有效盘符 .....	369	13.11.3	实现文件解压 .....	387
13.5.3	实现文件搜索功能 .....	370	13.12	文件分割与合并 .....	388
13.6	实现批量复制 .....	371	13.12.1	功能概述 .....	388
13.6.1	功能概述 .....	371	13.12.2	实现文件分割 .....	388
13.6.2	实现复制指定文件 .....	372	13.12.3	实现文件合并 .....	390
13.6.3	实现复制整个文件夹 .....	373	13.13	实现文件分类管理 .....	392
13.7	实现文件批量移动 .....	375	13.13.1	功能概述 .....	392
13.7.1	功能概述 .....	375	13.13.2	分类管理文件 .....	392
13.7.2	实现移动指定文件 .....	375			
第14章 图书管理模块 (Swing+SQL Server 2005 实现) .....					394
14.1	图书管理模块概述 .....	395	14.3.4	为窗体添加背景 .....	408
14.1.1	模块概述 .....	395	14.4	添加图书类别 .....	409
14.1.2	功能结构 .....	395	14.4.1	功能概述 .....	409
14.1.3	程序预览 .....	395	14.4.2	保存图书类别 .....	409
14.2	关键技术 .....	397	14.5	修改和删除图书类别 .....	410
14.2.1	连接和操作数据库 .....	397	14.5.1	功能概述 .....	410
14.2.2	MenuActions 类的编写 .....	399	14.5.2	修改图书类别 .....	411
14.2.3	限制文本框长度类的 编写 .....	403	14.5.3	删除图书类别 .....	413
14.2.4	描述组合框索引与 内容类的编写 .....	404	14.6	添加图书信息 .....	414
14.2.5	在 JLabel 上添加图片类 .....	406	14.6.1	功能概述 .....	414
14.3	主窗体 .....	406	14.6.2	保存图书信息 .....	414
14.3.1	功能概述 .....	406	14.7	修改和删除图书信息 .....	416
14.3.2	实现菜单栏的设计 .....	407	14.7.1	功能概述 .....	416
14.3.3	实现工具栏的设计 .....	408	14.7.2	修改图书信息 .....	416
			14.7.3	删除图书信息 .....	419
			14.8	查询图书信息 .....	420

14.8.1 功能概述 .....	420	14.8.3 显示全部图书信息 .....	422
14.8.2 查询满足条件的图书 .....	420		
<b>第 15 章 五子棋游戏模块 (Swing+Socket 网络技术实现) .....</b>	<b>424</b>		
15.1 五子棋模块概述 .....	425	15.4.1 功能概述 .....	437
15.1.1 模块概述 .....	425	15.4.2 聊天面板实现 .....	438
15.1.2 功能结构 .....	425	15.4.3 实现用户信息面板与 列表面板 .....	439
15.1.3 系统预览 .....	425		
15.2 关键技术 .....	428	15.5 下棋面板 .....	440
15.2.1 实现透明的登录界面 .....	428	15.5.1 功能概述 .....	440
15.2.2 监控网络连接状态 .....	428	15.5.2 实现广告标题栏 .....	440
15.2.3 绑定属性的 JavaBean .....	429	15.5.3 绘制下棋面板中的图片 .....	441
15.2.4 在棋盘上绘制棋子 .....	430	15.5.4 下棋前的准备工作 .....	442
15.2.5 实现动态调整棋盘大小 .....	431	15.5.5 游戏控制面板按钮事件 .....	443
15.2.6 游戏悔棋 .....	432		
15.2.7 游戏回放 .....	433	15.6 棋盘面板 .....	446
15.3 游戏登录界面 .....	434	15.6.1 功能概述 .....	446
15.3.1 功能概述 .....	434	15.6.2 绘制棋盘面板 .....	447
15.3.2 绘制登录界面背景 .....	434	15.6.3 实现游戏规则算法 .....	450
15.3.3 增加窗体控件 .....	435	15.6.4 编写棋盘模型 .....	453
15.3.4 处理“登录”按钮事件 .....	436	15.6.5 编写联机通信类 .....	454
15.4 游戏主窗体 .....	437		

## 第二篇 项目实战篇

<b>第 16 章 酒店管理系统 (Swing+SQL Server 2005 实现) .....</b>	<b>459</b>
16.1 开发背景 .....	460
16.2 系统分析 .....	460
16.3 系统设计 .....	461
16.3.1 系统目标 .....	461
16.3.2 系统功能结构 .....	461
16.3.3 系统预览 .....	461
16.3.4 文件夹结构设计 .....	463
16.3.5 业务流程图 .....	463
16.4 数据库设计 .....	464
16.4.1 数据库概要说明 .....	464
16.4.2 数据库概念设计 .....	464
16.4.3 数据库逻辑设计 .....	465
16.5 公共模块设计 .....	466
16.5.1 编写数据库连接类 .....	466
16.5.2 封装常用的操作数据库的 方法 .....	467
16.5.3 自定义表格控件 .....	469
16.5.4 编写利用正则表达式验证 数据合法性的方法 .....	470
16.6 主窗体模块设计 .....	470
16.6.1 主窗体模块概述 .....	470
16.6.2 主窗体模块技术分析 .....	471
16.6.3 主窗体模块实现过程 .....	471
16.7 用户登录窗口模块设计 .....	472
16.7.1 用户登录窗口模块概述 .....	472
16.7.2 用户登录窗口模块 技术分析 .....	473
16.7.3 用户登录窗口模块 实现过程 .....	474
16.8 开台签单工作区设计 .....	477
16.8.1 开台签单工作区 功能概述 .....	477

16.8.2	开台签单工作区		
	技术分析 .....	478	
16.8.3	开台签单工作区		
	实现过程 .....	479	
16.8.4	单元测试 .....	483	
16.9	自动结账工作区设计 .....	484	
16.9.1	自动结账工作区		
	功能概述 .....	484	
16.9.2	自动结账工作区		
	技术分析 .....	484	
16.9.3	自动结账工作区		
	实现过程 .....	485	
16.10	结账报表工作区设计 .....	486	
16.10.1	结账报表工作区		
	功能概述 .....	486	
16.10.2	结账报表工作区		
	技术分析 .....	487	
16.10.3	结账报表工作区		
	实现过程 .....	488	
16.10.4	单元测试 .....	491	
16.11	后台管理工作区设计 .....	492	
16.11.1	后台管理工作区		
	功能概述 .....	492	
16.11.2	后台管理工作区		
	技术分析 .....	493	
16.11.3	后台管理工作区		
	实现过程 .....	494	
16.11.4	单元测试 .....	500	
16.12	开发问题解析 .....	500	
<b>第 17 章 企业人事管理系统 (Swing+Hibernate+Oracle 实现) .....</b>		<b>503</b>	
17.1	开发背景 .....	504	
17.2	系统分析 .....	504	
17.3	系统设计 .....	504	
17.3.1	系统目标 .....	504	
17.3.2	系统功能结构 .....	504	
17.3.3	系统预览 .....	506	
17.3.4	业务流程图 .....	507	
17.3.5	文件夹结构设计 .....	507	
17.4	数据库设计 .....	508	
17.4.1	数据库分析 .....	508	
17.4.2	数据库概念设计 .....	508	
17.4.3	数据库逻辑结构设计 .....	509	
17.5	主窗体设计 .....	510	
17.5.1	导航栏的设计 .....	510	
17.5.2	工具栏的设计 .....	512	
17.6	公共模块设计 .....	514	
17.6.1	编写 Hibernate 配置文件 .....	514	
17.6.2	编写 Hibernate 持久化类和映射文件 .....	515	
17.6.3	编写通过 Hibernate 操作持久化对象的常用方法 .....	516	
17.6.4	创建用于特殊效果的部门树对话框 .....	517	
17.6.5	创建通过部门树选取员工的面板 and 对话框 .....	518	
17.7	人事管理模块设计 .....	520	
17.7.1	人事管理模块功能概述 .....	520	
17.7.2	人事管理模块技术分析 .....	521	
17.7.3	人事管理模块实现过程 .....	522	
17.7.4	单元测试 .....	526	
17.8	待遇管理模块设计 .....	527	
17.8.1	待遇管理模块功能概述 .....	527	
17.8.2	待遇管理模块技术分析 .....	527	
17.8.3	待遇管理模块实现过程 .....	528	
17.9	系统维护模块设计 .....	532	
17.9.1	系统维护模块功能概述 .....	532	
17.9.2	系统维护模块技术分析 .....	533	
17.9.3	系统维护模块实现过程 .....	534	
17.9.4	单元测试 .....	537	
17.10	开发问题解析 .....	538	
17.11	Hibernate 关联关系的建立方法 .....	538	
17.11.1	建立一对一关联 .....	538	
17.11.2	建立一对多关联 .....	539	

# 第一篇

## 典型模块篇

本篇主要内容：

- 第 1 章 备忘录模块
- 第 2 章 学生成绩管理模块
- 第 3 章 常用照片管理模块
- 第 4 章 定制打印模块
- 第 5 章 短信收发模块
- 第 6 章 FTP 上传下载模块
- 第 7 章 局域网通信模块
- 第 8 章 区域地图模块
- 第 9 章 序列号注册模块
- 第 10 章 PDF 查看模块
- 第 11 章 动态考题模块
- 第 12 章 多功能查询模块
- 第 13 章 文件分割模块
- 第 14 章 图书管理模块
- 第 15 章 五子棋游戏模块

书山有路  
勤为径  
学海无涯  
苦作舟



# 第 1 章

---

## 备忘录模块

( Swing+JDBC 实现 )

随着人们生活节奏的加快，每人每天都需要完成很多事情。为了避免遗漏某些重要事件，可以将其保存在备忘录中。备忘录的形式很灵活，随着电脑的普及，越来越多的用户选择备忘录软件。本章将开发一个备忘录模块，读者可以将其嵌入到自己的系统中。通过本章的学习，读者能够学到：

- » 使用 Swing 技术开发桌面程序
- » 绘制艺术文字
- » 使用 JDBC 操作数据库
- » 使用正则表达式校验日期
- » 调用系统小工具

## 1.1 备忘录模块概述

### 1.1.1 模块概述

在日常生活中,经常有些重要的事情需要记住,通常,人们喜欢将其记录到纸质备忘录中,本章将开发一个备忘录模块,它包括了备忘录的添加、修改和删除功能,读者可以将其作为一个子功能嵌入到自己开发的系统中。

### 1.1.2 功能结构

备忘录软件的主要功能包括添加备忘录、修改备忘录、查询备忘录、删除备忘录等。此外,还能够打开 Windows 系统中的记事本、计算器等小工具。其功能结构如图 1.1 所示。

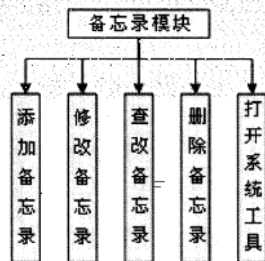


图 1.1 备忘录模块功能结构图

### 1.1.3 程序预览

备忘录模块由多个窗体组成,主窗体的运行效果如图 1.2 所示,主要功能是调用执行本系统的所有功能。

“增加备忘录”窗体的运行效果如图 1.3 所示,主要功能是完成备忘录的增加。这里所有内容都是必须填写的,其中时间的格式要求是“2010-12-22”的样式,否则将不能增加备忘录。

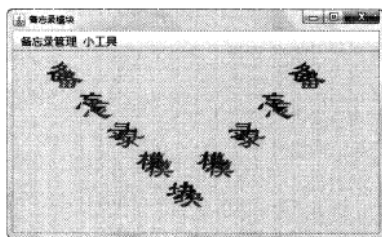


图 1.2 “备忘录模块”主窗体

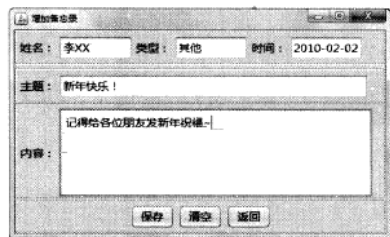


图 1.3 “增加备忘录”窗体

“修改备忘录”窗体的运行效果如图 1.4 所示,主要功能是完成对已经增加的备忘录的修改。修改时也要注意保证信息的完整性及日期的格式。

“查询备忘录”窗体的运行效果如图 1.5 所示,主要功能是对已经保存的备忘录进行查询。

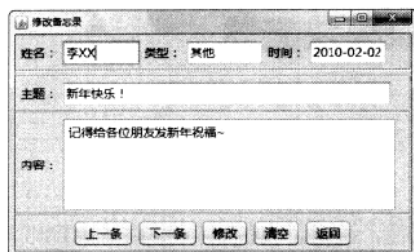


图 1.4 “修改备忘录”窗体

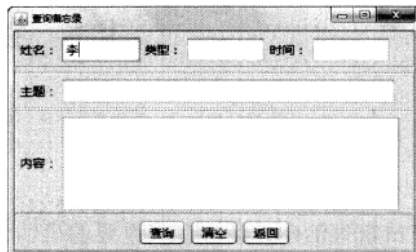


图 1.5 “查询备忘录”窗体

查询的条件是任意的，而且支持模糊查询。如果没有找到符合条件的记录，则进行提示，如图 1.6 所示。

否则直接显示查询的结果，如图 1.7 所示。如果存在多个查询结果，则可以使用“上一条”和“下一条”按钮进行查看。



图 1.6 查询失败时的窗体

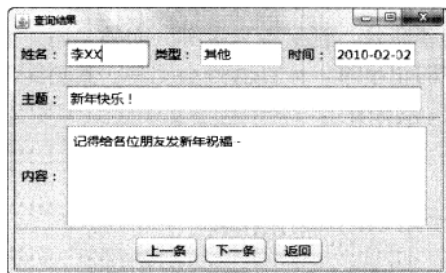


图 1.7 “查询结果”窗体



图 1.8 “删除备忘录”窗体

删除备忘录窗体如图 1.8 所示，它用于删除当前窗体中显示的备忘录信息。可以使用“上一条”和“下一条”按钮查看其他的信息。

## 1.2 关键技术



4

### 1.2.1 绘制艺术字



在程序的主窗体中，显示的彩色文本信息“备忘录模块”是使用 Java 的绘图技术绘制在面板控件上的，其关键代码如下：

```
public void paint(Graphics g) {
```





```

super.paint(g); // 调用父类构造方法
Graphics2D g2 = (Graphics2D) g; // 获得 Graphics2D 对象
String str = "备忘录模块"; // 设置要显示的字符串
Font font = new Font("隶书", Font.BOLD | Font.ITALIC, 40); // 创建新字体
g2.setFont(font); // 应用字体
for (int i = 0; i < str.length(); i++) {
    g2.setColor(Color.GRAY); // 设置前景色
    g2.drawString(str.charAt(i) + "", 50 + i * font.getSize(), 50 + i *
        font.getSize()); // 在指定位置绘制文本
    g2.drawString(str.charAt(i) + "", 370 - i * font.getSize(), 50 + i *
        font.getSize()); // 在指定位置绘制文本
}
for (int i = 0; i < str.length(); i++) {
    g2.setColor(Color.MAGENTA); // 设置前景色
    g2.drawString(str.charAt(i) + "", 40 + i * font.getSize(), 40 + i *
        font.getSize()); // 在指定位置绘制文本
    g2.drawString(str.charAt(i) + "", 360 - i * font.getSize(), 40 + i *
        font.getSize()); // 在指定位置绘制文本
}
}

```

这里主要使用 Graphics2D 类中的 drawString 方法来完成字符串的绘制, 该方法的声明如下:

```
public abstract void drawString(String str,int x,int y)
```

- str: 需要绘制的字符串。
- x: 绘制的字符串位置的 x 坐标。
- y: 绘制的字符串位置的 y 坐标。

## 1.2.2 窗体居中显示

为了让窗体看起来更加舒适, 可以设置窗体显示位置为居中显示。这就需要获得用户显示器的大小并根据窗体大小来计算显示位置, 这是通过工具类 WindowUtil 实现的, 该类的代码如下:

```

public class WindowUtil {
    // 将窗体大小设置成 500×309 像素
    public static Dimension getSize() {
        return new Dimension(500, 309);
    }
    // 计算窗体居中显示时左上角的坐标
    public static Point getLocation() {
        Toolkit toolKit = Toolkit.getDefaultToolkit(); // 获得 Toolkit 实例
        Dimension screenSize = toolKit.getScreenSize(); // 获得显示器大小
        if ((screenSize.width < getSize().width) || (screenSize.height <
            getSize().height)) {
            JOptionPane.showMessageDialog(null, "显示器分辨率至少为 500×309", "",
                JOptionPane.WARNING_MESSAGE);
            System.exit(0); // 终止程序
        }
        int x = (screenSize.width - getSize().width) / 2; // 计算左上角横坐标
        int y = (screenSize.height - getSize().height) / 2; // 计算左上角纵坐标
        return new Point(x, y);
    }
}

```

首先定义 getSize() 方法, 它返回窗体的大小, 这里将窗体的宽度设置为 500 像素, 高度设置为 309 像素。Dimension 类用于封装控件的宽度和高度, 使用该类有利于以面向对





象的方式编写代码。

接着定义 `getLocation()` 方法，它返回窗体左上角的坐标。这里首先获得 `Toolkit` 类的对象，由于该类是抽象类，不能直接使用 `new` 操作符实例化，因此使用该类定义的 `getDefaultToolkit()` 方法获得实例。该方法的声明如下：

```
public static Toolkit getDefaultToolkit()
```

`Toolkit` 类中的 `getScreenSize()` 方法可以获得用户显示器的分辨率，该方法的声明如下：

```
public abstract Dimension getScreenSize() throws HeadlessException
```

如果用户的分辨率小于  $500 \times 309$  像素，则弹出提示对话框并终止程序。

### 1.2.3 使用 JavaBean 封装信息

在本模块中，用户需要填写备忘录，这涉及了大量的信息，例如姓名、主题、内容、类型和时间等。通常定义一个类来保存这些信息，将它们设置成私有字段，并提供对应的 `get` 和 `set` 方法，这样就构成了一个 `JavaBean`。`MemoBean` 的代码如下：

```
public class MemoBean {  
    private int id; // 备忘录编号  
    private String username; // 用户名称  
    private String title; // 备忘录标题  
    private String content; // 备忘录内容  
    private String memotype; // 备忘录类型  
    private String memotime; // 备忘录时间  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getUsername() {  
        return username;  
    }  
    public void setUsername(String username) {  
        this.username = username;  
    }  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
    public String getContent() {  
        return content;  
    }  
    public void setContent(String content) {  
        this.content = content;  
    }  
    public String getMemotype() {  
        return memotype;  
    }  
    public void setMemotype(String memotype) {  
        this.memotype = memotype;  
    }  
    public String getMemotime() {  
        return memotime;  
    }  
    public void setMemotime(String memotime) {
```







```

        this.memotime = memotime;
    }
}

```

## 1.2.4 获得 MySQL 数据库连接

本程序的后台使用 MySQL 数据库来保存备忘录信息, 在使用 JDBC 操作数据库时, 需要先获得数据库连接对象, 这是使用 JdbcHelper 类的 getConnection() 方法实现的, 该方法的代码如下:

```

private static Connection getConnection() {
    Connection conn = null;
    try {
        Class.forName(DRIVER);           // 加载数据库驱动
        // 获得数据库连接
        conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        return conn;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

```

方法中用到的 DRIVER、URL、USERNAME 和 PASSWORD 是字符串常量, 定义在 JdbcConfig 接口中, 该接口的代码如下:

```

public interface JdbcConfig {
    String DRIVER = "com.mysql.jdbc.Driver";
    String URL = "jdbc:mysql://localhost:3306/db_memomodule";
    String USERNAME = "root";
    String PASSWORD = "111";
}

```



**注意** 在使用 JDBC 操作数据库时需要为项目添加数据库驱动文件。

## 1.2.5 批量处理数据库操作

在处理大量数据时, 使用批处理操作能够大幅度提高效率, 这里使用该技术完成备忘录的保存和修改操作。

### 1. 保存备忘录信息

使用 JdbcHelper 类的 save() 方法来保存备忘录信息, 其关键代码如下:

```

public static int save(MemoBean memo) {
    String sql = "insert into tb_memo (username, title, content, memotype, memotime) values (?, ?, ?, ?, ?)";
    Connection conn = getConnection();
    PreparedStatement ps = null;
    try {
        ps = conn.prepareStatement(sql);
        ps.setString(1, memo.getUsername());
    }
}

```





```
        ps.setString(2, memo.getTitle());
        ps.setString(3, memo.getContent());
        ps.setString(4, memo.getMemotype());
        ps.setString(5, memo.getMemotime());
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return -1;
}
```

使用 PreparedStatement 接口可以完成批处理操作。批处理时使用问号来表示需要添加的变量，然后使用该接口中与变量类型对应的 set()方法来添加变量。例如 setString()表示设置字符串的值，这个方法的第一个参数表示变量的位置，第二个参数表示变量的值。最后调用 executeUpdate()方法来完成批量保存。

## 2. 修改备忘录

修改备忘录使用的是 JdbcHelper 类中的 update()方法，其关键代码如下：

```
public static int update(MemoBean memo) {
    String sql = "update tb_memo set username = ?, title = ?, content = ?, memotype = ?, memotime = ? where id = ?";
    Connection conn = getConnection();
    PreparedStatement ps = null;
    try {
        ps = conn.prepareStatement(sql);
        ps.setString(1, memo.getUsername());
        ps.setString(2, memo.getTitle());
        ps.setString(3, memo.getContent());
        ps.setString(4, memo.getMemotype());
        ps.setString(5, memo.getMemotime());
        ps.setInt(6, memo.getId());
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```







```

    }
    return -1;
}

```

## 1.2.6 使用 List 保存查询结果

在 JDBC 中, 查询操作的返回值是 `ResultSet` 类型的对象, 它使用起来并不方便, 因此通常将结果保存到 `List` 中。

### 1. 保存全部备忘录信息

本模块使用 `JdbcHelper` 类的 `queryAll()` 方法保存表格中的全部数据, 其关键代码如下:

```

public static List<MemoBean> queryAll() {
    String sql = "select * from tb_memo;";
    List<MemoBean> results = new ArrayList<MemoBean>();
    Connection conn = getConnection();
    Statement stat = null;
    ResultSet rs = null;
    try {
        stat = conn.createStatement();
        rs = stat.executeQuery(sql);
        while (rs.next()) {
            MemoBean memo = new MemoBean();
            memo.setId(rs.getInt("id"));
            memo.setUsername(rs.getString("username"));
            memo.setTitle(rs.getString("title"));
            memo.setContent(rs.getString("content"));
            memo.setMemotype(rs.getString("memotype"));
            memo.setMemotime(rs.getString("memotime"));
            results.add(memo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        // 释放资源
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return results;
}

```







## 2. 保存部分备忘录信息

本模块使用 JdbcHelper 类的 query () 方法来保存满足查询条件的全部数据, 这里将查询条件封装到 MemoBean 对象中, 其关键代码如下:

```
public static List<MemoBean> query(MemoBean memo) {
    String username = memo.getUsername();
    String title = memo.getTitle();
    String content = memo.getContent();
    String memotype = memo.getMemotype();
    String memotime = memo.getMemotime();

    StringBuilder sql = new StringBuilder("select * from tb_memo where 1=1 ");
    if (!username.isEmpty()) {
        sql.append(" and username like '%" + username + "%' ");
    }
    if (!title.isEmpty()) {
        sql.append(" and title like '%" + title + "%' ");
    }
    if (!content.isEmpty()) {
        sql.append(" and content like '%" + content + "%' ");
    }
    if (!memotype.isEmpty()) {
        sql.append(" and memotype like '%" + memotype + "%' ");
    }
    if (!memotime.isEmpty()) {
        sql.append(" and memotime like '%" + memotime + "%' ");
    }
    sql.append(";");
    List<MemoBean> results = new ArrayList<MemoBean>();
    Connection conn = getConnection();
    Statement stat = null;
    ResultSet rs = null;
    try {
        stat = conn.createStatement();
        rs = stat.executeQuery(sql.toString());
        while (rs.next()) {
            MemoBean tempMemo = new MemoBean();
            tempMemo.setId(rs.getInt("id"));
            tempMemo.setUsername(rs.getString("username"));
            tempMemo.setTitle(rs.getString("title"));
            tempMemo.setContent(rs.getString("content"));
            tempMemo.setMemotype(rs.getString("memotype"));
            tempMemo.setMemotime(rs.getString("memotime"));
            results.add(tempMemo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            }
        }
    }
}
```

// 释放资源







```

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return results;
}

```

### 1.2.7 使用正则表达式校验日期

在输入时间时, 需要用户按照类似“2010-12-22”的样式输入, 可以使用正则表达式完成输入格式的校验工作, 但是非常复杂。这里使用 Java 代码来辅助校验, 其关键代码如下:

```

public static boolean validateTimeFormat(String memotime) {
    String regx = "([123]\\d{3}-([0][1-9])|([1][0-2]))-([0][1-9]|([12]\\d)|([3][01]))";
    Pattern pattern = Pattern.compile(regx);
    Matcher matcher = pattern.matcher(memotime);
    if (!matcher.matches()) {
        return false;
    } else {
        String[] times = memotime.split("-");
        Integer year = Integer.parseInt(times[0]); // 将输入的时间用-分割
        Integer month = Integer.parseInt(times[1]); // 获得用户输入的年
        Integer day = Integer.parseInt(times[2]); // 获得用户输入的月
        GregorianCalendar calendar = new GregorianCalendar(); // 获得用户输入的日
        // 闰年 2 月天数不能大于 29
        if (calendar.isLeapYear(year) && (month == 2) && (day > 29)) {
            return false;
        } else if ((!calendar.isLeapYear(year)) && (month == 2) && (day > 28)) {
            // 非闰年 2 月天数不能大于 28
            return false;
        } else {
            switch (month) {
                case 4: // 4、6、9、11 月都是 30 天
                case 6:
                case 9:
                case 11:
                    if (day > 30) {
                        return false;
                    } else {
                        return true;
                    }
                default:
                    return true;
            }
        }
    }
}

```

字符串“([123]\\d{3}-([0][1-9])|([1][0-2]))-([0][1-9]|([12]\\d)|([3][01]))”是自定义的正则表达式, [123]表示 1、2 和 3 三个数字选取一个; (\\d)表示 0~9 中任意一个数字; {3}





表示将前面临近的表达式重复 3 遍，即`(\\d){3}`与`(\\d)(\\d)(\\d)`等价。对于年，其取值范围设置成了 1000~3999。对于月，合法的格式应该是 01~09 和 10、11、12。使用`[0][1-9]`来表示 01~09，使用`[1][0-2]`来表示 10、11、12。中间的`|`表示或。对于日，合法的格式应该是 01~09、10~29、30、31，使用`[0][0-9]`表示 01~09，使用`[12]\\d`表示 10~29，使用`[3][01]`表示 30 和 31。

但是这里并没有对具体的年份、月份进行校验。例如闰年的 2 月不能有 30 天、非闰年的 2 月不能有 29 天。每年的 4、6、9、11 月都是 30 天，不能有 31 天。使用正则表达式完成这类校验非常复杂，因此使用 Java 代码来完成。

`Pattern` 类是在 `java.util.regex` 包中定义的正则表达式类，它的 `compile()` 方法能够解析给定的正则表达式，该方法的声明如下：

```
public static Pattern compile(String regex)
```

regex: 需要解析的正则表达式。

`Matcher` 类的 `matches()` 方法能够判断给定的字符串是否符合正则表达式，该方法的声明如下：

```
public boolean matches()
```

### 1.2.8 调用系统工具

在本模块中，提供了系统工具的调用功能，用户可以打开 Windows 系统中的记事本和计算器程序，这是使用 `Runtime` 类完成的，打开记事本的关键代码如下：

```
Runtime runTime = Runtime.getRuntime();           // 获得 Runtime 类型对象
try {
    runTime.exec("notepad");                       // 执行 notepad 命令打开记事本
} catch (IOException e1) {
    e1.printStackTrace();
}
```

`Runtime` 类的 `exec()` 方法可以执行系统命令，该方法的声明如下：

```
public Process exec(String command) throws IOException
```

command: 本地系统命令。



本程序是在 Windows 系统中开发的，并不适用于 Linux 系统。

## 1.3 主窗体



12

### 1.3.1 功能概述



备忘录模块由多个窗体组成，主窗体的运行效果如图 1.9 所示，主要功能是调用执行本系统的所有功能。



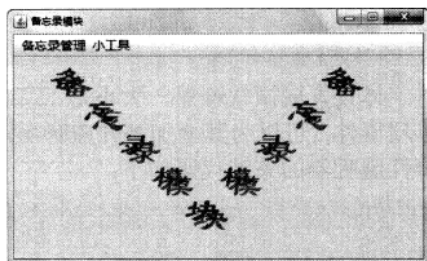


图 1.9 备忘录模块主窗体

### 1.3.2 添加菜单及菜单项

在主窗体中，使用菜单来管理各个功能，下面介绍菜单及菜单项的使用。

在窗体中添加菜单前，需要先添加菜单栏，即 `JMenuBar`，其关键代码如下：

```
JMenuBar menuBar = new JMenuBar();           // 创建菜单条
setJMenuBar(menuBar);                         // 应用菜单条
```

接着向菜单栏中增加菜单，其关键代码如下：

```
JMenu memoManagementMenu = new JMenu("备忘录管理");           // 创建菜单
memoManagementMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
menuBar.add(memoManagementMenu);                               // 应用菜单

JMenu toolMenu = new JMenu("小工具");                           // 创建菜单
toolMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15));         // 设置字体
menuBar.add(toolMenu);                                           // 应用菜单
```

这里创建了“备忘录管理”和“小工具”两个菜单。

下面为这两个菜单分别增加菜单项，其关键代码如下：

```
JMenuItem addMemoMenuItem = new JMenuItem("增加备忘录"); // 创建菜单项
// 省略事件处理相关代码
addMemoMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
memoManagementMenu.add(addMemoMenuItem);                       // 应用菜单项

JMenuItem modifyMemoMenuItem = new JMenuItem("修改备忘录"); // 创建菜单项
// 省略事件处理相关代码
modifyMemoMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
memoManagementMenu.add(modifyMemoMenuItem);                   // 应用菜单项

JMenuItem queryMemoMenuItem = new JMenuItem("查询备忘录"); // 创建菜单项
// 省略事件处理相关代码
queryMemoMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
memoManagementMenu.add(queryMemoMenuItem);                   // 应用菜单项

JMenuItem deleteMemoMenuItem = new JMenuItem("删除备忘录"); // 创建菜单项
// 省略事件处理相关代码
deleteMemoMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
memoManagementMenu.add(deleteMemoMenuItem);                   // 应用菜单项

JMenuItem notepadMenuItem = new JMenuItem("记事本");         // 创建菜单项
// 省略事件处理相关代码
notepadMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
toolMenu.add(notepadMenuItem);                               // 应用菜单项

JMenuItem calculatorMenuItem = new JMenuItem("计算器");       // 创建菜单项
```





```
// 省略事件处理相关代码
calculatorMemoMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
toolMenu.add(calculatorMemoMenuItem); // 应用菜单项
```

在上述代码中，统一将字体设置成微软雅黑，大小是 15 磅。

为了处理用户单击菜单项事件，可以为其增加事件监听器，下面以“增加备忘录”菜单项为例进行讲解。添加事件监听器的关键代码如下：

```
addMemoMenuItem.addActionListener(new ActionListener() { // 监听菜单项事件
    @Override
    public void actionPerformed(ActionEvent e) {
        do_addMemoMenuItem_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_addMemoMenuItem_actionPerformed()` 方法，它是由 IDE 工具自动生成的，用于创建 `MemoAdditionFrame` 并将其设置成可见，其关键代码如下：

```
protected void do_addMemoMenuItem_actionPerformed(ActionEvent e) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                MemoAdditionFrame frame = new MemoAdditionFrame();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

### 1.3.3 绘制窗体中的艺术字

为了美化程序，这里采用绘制艺术字的方式，它是通过重写 `JPanel` 控件的 `paint()` 方法完成的，其关键代码如下：

```
contentPane = new JPanel() {
    private static final long serialVersionUID = 2179076594345705736L;

    @Override
    public void paint(Graphics g) {
        super.paint(g); // 调用父类构造方法
        Graphics2D g2 = (Graphics2D) g; // 获得 Graphics2D 对象
        String str = "备忘录模块"; // 设置要显示的字符串
        Font font = new Font("隶书", Font.BOLD | Font.ITALIC, 40); // 创建新字体
        g2.setFont(font); // 应用字体
        for (int i = 0; i < str.length(); i++) {
            g2.setColor(Color.GRAY); // 设置前景色
            g2.drawString(str.charAt(i) + "", 50 + i * font.getSize(), 50 + i * font.getSize()); // 在指定位置绘制文本
            g2.drawString(str.charAt(i) + "", 370 - i * font.getSize(), 50 + i * font.getSize()); // 在指定位置绘制文本
        }
        for (int i = 0; i < str.length(); i++) {
            g2.setColor(Color.MAGENTA); // 设置前景色
            g2.drawString(str.charAt(i) + "", 40 + i * font.getSize(), 40 + i * font.getSize()); // 在指定位置绘制文本
            g2.drawString(str.charAt(i) + "", 360 - i * font.getSize(), 40 + i * font.getSize()); // 在指定位置绘制文本
        }
    }
}
```







```
};
```

### 1.3.4 设置窗体显示位置和大小

在使用窗体之前,需要为其设置显示位置和大小。因为 Swing 中窗体的默认大小是 0×0。这里使用自定义的 WindowUtil 类,其关键代码如下:

```
setLocation(WindowUtil.getLocation()); // 设置窗体显示位置
setSize(WindowUtil.getSize()); // 设置窗体大小
```



对于其他的窗体也采用类似的方式设置显示位置和大小,下面省略这部分

的讲解。

至此完成了 MainFrame 类的编写,关于该类的详细代码请读者参考源文件。

## 1.4 增加备忘录

### 1.4.1 功能概述

增加备忘录窗体用于增加备忘录,包括姓名、主题、内容等相关信息。单击主窗体“备忘录管理”/“增加备忘录”菜单项,就可以打开“增加备忘录”窗体,如图 1.10 所示。

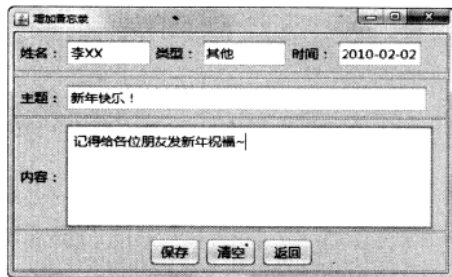


图 1.10 增加备忘录窗体

### 1.4.2 设置文本框和文本区控件

在图 1.10 中,定义了很多文本框和文本区控件,用于接收用户输入的信息,并且对不同的面板使用了不同的布局管理器。为了方便用户,使用边框将信息进行分类,其关键代码如下:

```
JPanel othersPanel = new JPanel(); // 创建面板
contentPane.add(othersPanel, BorderLayout.NORTH); // 应用面板
othersPanel.setLayout(new GridLayout(2, 1, 5, 5)); // 为面板设置网格布局

JPanel nttPanel = new JPanel(); // 创建面板
// 设置面板的边框
nttPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_nttPanel = (FlowLayout) nttPanel.getLayout(); // 获得面板布局
fl_nttPanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
othersPanel.add(nttPanel); // 应用面板
```





```
JLabel usernameLabel = new JLabel("姓名: "); // 创建标签
usernameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameLabel); // 应用标签

usernameTextField = new JTextField(); // 创建文本域
usernameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameTextField); // 应用文本域
usernameTextField.setColumns(6); // 设置文本域宽度

JLabel memotypeLabel = new JLabel("类型: "); // 创建标签
memotypeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeLabel); // 应用标签

memotypeTextField = new JTextField(); // 创建文本域
memotypeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeTextField); // 应用文本域
memotypeTextField.setColumns(6); // 设置文本域宽度

JLabel memotimeLabel = new JLabel("时间: "); // 创建标签
memotimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeLabel); // 应用标签

memotimeTextField = new JTextField(); // 创建文本域
memotimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeTextField); // 应用文本域
memotimeTextField.setColumns(6); // 设置文本域宽度

JPanel titlePanel = new JPanel(); // 创建面板
titlePanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 设置面板的边框
FlowLayout fl_titlePanel = (FlowLayout) titlePanel.getLayout(); // 获得面板布局
fl_titlePanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
othersPanel.add(titlePanel); // 应用面板

JLabel titleLabel = new JLabel("主题: "); // 创建标签
titleLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titleLabel.add(titleLabel); // 应用标签

titleTextField = new JTextField(); // 创建文本域
titleTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titleLabel.add(titleTextField); // 应用文本域
titleTextField.setColumns(28); // 设置文本域宽度

JPanel contentPanel = new JPanel(); // 创建面板
// 设置面板的边框
contentPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 获得面板布局
FlowLayout fl_contentPanel = (FlowLayout) contentPanel.getLayout();
fl_contentPanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
contentPanel.add(contentPanel, BorderLayout.CENTER); // 应用面板

JLabel contentLabel = new JLabel("内容: "); // 创建标签
contentLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
contentPanel.add(contentLabel); // 应用标签

contentTextArea = new JTextArea(); // 创建文本区
contentTextArea.setColumns(28); // 设置文本区列数
contentTextArea.setLineWrap(true); // 设置文本区自动换行
contentTextArea.setRows(5); // 设置文本区行数
contentTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
// 使用文本区创建滚动窗格
JScrollPane contentScrollPane = new JScrollPane(contentTextArea);
contentPanel.add(contentScrollPane); // 应用滚动窗格
```





### 1.4.3 添加工具按钮

在窗体中使用了“保存”、“清空”和“返回”3个按钮，并将它们放置在一个面板中，其关键的代码如下：

```
JPanel buttonPanel = new JPanel();           // 创建面板
// 设置面板的边框
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用面板

JButton saveButton = new JButton("保存");      // 创建“保存”按钮
// 省略事件处理相关代码
saveButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(saveButton);                  // 应用按钮

JButton clearButton = new JButton("清空");     // 创建“清空”按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton);                 // 应用按钮

JButton returnButton = new JButton("返回");    // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton);               // 应用按钮
```

### 1.4.4 保存备忘录信息

通过监听“保存”按钮单击事件，完成对用户增加的备忘录信息的保存功能。由于备忘录上的信息都非常重要，因此不能为空值。“保存”按钮事件监听器的关键代码如下：

```
saveButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_saveButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_saveButton_actionPerformed()` 方法，它是 IDE 工具自动生成的方法，用于对文本域和文本区的非空校验及保存用户输入信息，其关键代码如下：

```
protected void do_saveButton_actionPerformed(ActionEvent e) {
    String username = usernameTextField.getText().trim();
    if (username.isEmpty()) {
        JOptionPane.showMessageDialog(this, "姓名不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String title = titleTextField.getText().trim();
    if (title.isEmpty()) {
        JOptionPane.showMessageDialog(this, "主题不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String content = contentTextArea.getText().trim();
    if (content.isEmpty()) {
        JOptionPane.showMessageDialog(this, "内容不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
}
```





```
String memotype = memotypeTextField.getText().trim();
if (memotype.isEmpty()) {
    JOptionPane.showMessageDialog(this, "类型不能为空!", "", JOptionPane.
        WARNING_MESSAGE);
    return;
}
String memotime = memotimeTextField.getText().trim();
if (memotime.isEmpty()) {
    JOptionPane.showMessageDialog(this, "时间不能为空!", "", JOptionPane.
        WARNING_MESSAGE);
    return;
}
if (!ValidationUtil.validateTimeFormat(memotime)) {
    JOptionPane.showMessageDialog(this, "日期格式为 2010-12-22 样式!", "",
        JOptionPane.WARNING_MESSAGE);
    return;
}
MemoBean memo = new MemoBean(); // 创建 MemoBean
memo.setUsername(username);
memo.setTitle(title);
memo.setContent(content);
memo.setMemotype(memotype);
memo.setMemotime(memotime);
int result = JdbcHelper.save(memo); // 保存备忘录信息
if (result >= 0) {
    JOptionPane.showMessageDialog(this, "备忘录添加成功!");
    return;
} else {
    JOptionPane.showMessageDialog(this, "备忘录添加失败!");
    return;
}
}
```

### 1.4.5 清空备忘录信息

在增加完备忘录之后，可以使用“清空”按钮清除备忘录中的信息，其事件监听器关键代码如下：

```
clearButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_clearButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_clearButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，其关键代码如下：

```
protected void do_clearButton_actionPerformed(ActionEvent e) {
    usernameTextField.setText("");
    titleTextField.setText("");
    contentTextArea.setText("");
    memotypeTextField.setText("");
    memotimeTextField.setText("");
}
```







### 1.4.6 销毁窗体

在使用完该窗体之后，可以单击“返回”按钮销毁该窗体，其事件监听器关键代码如下：

```
returnButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_returnButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_returnButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，其关键代码如下：

```
protected void do_returnButton_actionPerformed(ActionEvent e) {
    dispose();
}
```

`dispose()` 方法可以释放该窗体及其子控件占用的资源，该方法的声明如下：

```
public void dispose()
```

## 1.5 修改备忘录

### 1.5.1 功能概述

修改备忘录窗体用于备忘录的浏览和修改。通过单击该窗体上的“上一条”和“下一条”按钮可以浏览信息。输入修改后的内容，单击“修改”按钮可以保存修改的信息。单击主窗体“备忘录管理”/“修改备忘录”菜单项，就可以打开修改备忘录窗体，如图 1.11 所示。



图 1.11 修改备忘录窗体

### 1.5.2 设置文本框和文本区控件

在图 1.11 中，定义了很多文本框和文本区控件用于接收用户输入的信息，并且对不同的面板使用了不同的布局管理器。为了方便用户，使用边框将信息进行分类，其关键代





码如下:

```

JPanel othersPanel = new JPanel();           // 创建面板
contentPane.add(othersPanel, BorderLayout.NORTH); // 应用面板
othersPanel.setLayout(new GridLayout(2, 1, 5, 5)); // 为面板设置网格布局

JPanel nttPanel = new JPanel();               // 创建面板
// 设置面板的边框
nttPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_nttPanel = (FlowLayout) nttPanel.getLayout(); // 获得面板布局
fl_nttPanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
othersPanel.add(nttPanel); // 应用面板

JLabel usernameLabel = new JLabel("姓名:"); // 创建标签
usernameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameLabel); // 应用标签

usernameTextField = new JTextField(); // 创建文本域
usernameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameTextField); // 应用文本域
usernameTextField.setColumns(6); // 设置文本域宽度

JLabel memotypeLabel = new JLabel("类型:"); // 创建标签
memotypeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeLabel); // 应用标签

memotypeTextField = new JTextField(); // 创建文本域
memotypeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeTextField); // 应用文本域
memotypeTextField.setColumns(6); // 设置文本域宽度

JLabel memotimeLabel = new JLabel("时间:"); // 创建标签
memotimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeLabel); // 应用标签

memotimeTextField = new JTextField(); // 创建文本域
memotimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeTextField); // 应用文本域
memotimeTextField.setColumns(6); // 设置文本域宽度

JPanel titlePanel = new JPanel(); // 创建面板
// 设置面板的边框
titlePanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_titlePanel = (FlowLayout) titlePanel.getLayout(); // 获得面板布局
fl_titlePanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
othersPanel.add(titlePanel); // 应用面板

JLabel titleLabel = new JLabel("主题:"); // 创建标签
titleLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleLabel); // 应用标签

titleTextField = new JTextField(); // 创建文本域
titleTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleTextField); // 应用文本域
titleTextField.setColumns(28); // 设置文本域宽度

JPanel contentPanel = new JPanel(); // 创建面板
// 设置面板的边框
contentPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_contentPanel = (FlowLayout) contentPanel.getLayout(); // 获得面板布局
fl_contentPanel.setAlignment(FlowLayout.LEFT); // 面板中控件采用左对齐
contentPane.add(contentPanel, BorderLayout.CENTER); // 应用面板

JLabel contentLabel = new JLabel("内容:"); // 创建标签
contentLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体

```







```

contentPanel.add(contentLabel); // 应用标签

contentTextArea = new JTextArea(); // 创建文本区
contentTextArea.setColumns(28); // 设置文本区列数
contentTextArea.setLineWrap(true); // 设置文本区自动换行
contentTextArea.setRows(5); // 设置文本区行数
contentTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
// 使用文本区创建滚动窗格
JScrollPane contentScrollPane = new JScrollPane(contentTextArea);
contentPanel.add(contentScrollPane); // 应用滚动窗格

```

### 1.5.3 添加工具按钮

在窗体中使用了“上一条”、“下一条”、“修改”、“清空”和“返回”5个按钮，并将它们放置在一个面板中，其关键的代码如下：

```

JPanel buttonPanel = new JPanel(); // 创建面板
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 设置面板的边框
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用面板

JButton previousButton = new JButton("上一条"); // 创建“上一条”按钮
// 省略事件处理相关代码
previousButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(previousButton); // 应用按钮

JButton nextButton = new JButton("下一条"); // 创建“下一条”按钮
// 省略事件处理相关代码
nextButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(nextButton); // 应用按钮

JButton modifyButton = new JButton("修改"); // 创建“修改”按钮
// 省略事件处理相关代码
modifyButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(modifyButton); // 应用按钮

JButton clearButton = new JButton("清空"); // 创建“清空”按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton); // 应用按钮

```

### 1.5.4 填充备忘录信息

修改是在原来信息基础上进行的修改，因此需要先获取数据库中的备忘录信息并将其添加到备忘录中，这里使用了 JdbcHelper 类中的 queryAll() 方法获得所有的备忘录信息，然后使用自定义的 updateContent() 方法根据索引值添加备忘录信息到窗体。该方法的关键代码如下：

```

private void updateContent(int index) {
    MemoBean memo = results.get(index); // 获得第 index 条备忘录信息
    usernameTextField.setText(memo.getUsername());
    titleTextField.setText(memo.getTitle());
}

```







```
contentTextArea.setText(memo.getContent());
memotypeTextField.setText(memo.getMemotype());
memotimeTextField.setText(memo.getMemotime());
}
```

### 1.5.5 修改前一条备忘录信息

单击“上一条”按钮，将显示上一条备忘录的信息，用户可以在此基础上进行修改。在 MemoModificationFrame 类中，定义的该按钮事件监听器关键代码如下：

```
previousButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_previousButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 do\_previousButton\_actionPerformed() 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是第一条记录，则提示用户，其关键代码如下：

```
protected void do_previousButton_actionPerformed(ActionEvent e) {
    if (index <= 0) {
        JOptionPane.showMessageDialog(this, "已经是第一条记录!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    updateContent(--index);           //显示上一条信息
}
```



说明 index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

### 1.5.6 修改后一条备忘录信息

单击“下一条”按钮，将显示下一条备忘录的信息，用户可以在此基础上进行修改。在 MemoModificationFrame 类中，定义的事件监听器关键代码如下：

```
nextButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_nextButton_actionPerformed(e);
    }
});
```



在事件监听器中，调用了 do\_nextButton\_actionPerformed() 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是最后一条记录，则提示用户，其关键代码如下：

```
protected void do_nextButton_actionPerformed(ActionEvent e) {
    if (index >= (results.size() - 1)) {
        JOptionPane.showMessageDialog(this, "已经是最后一条记录!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
}
```



```

    }
    updateContent(++index);                //显示下一条信息
}

```



说明 index 是指 List 接口中元素的索引, 因此是从 0 开始计数的。

### 1.5.7 修改备忘录信息

通过监听“修改”按钮单击事件, 完成对用户修改的备忘录信息的保存功能。由于备忘录中的信息都非常重要, 因此不能为空值。“修改”按钮事件监听器关键代码如下:

```

modifyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_modifyButton_actionPerformed(e);
    }
});

```

do\_modifyButton\_actionPerformed()方法是 IDE 工具生产的方法, 它完成了对文本域和文本区的非空校验及保存用户修改信息的功能, 其关键代码如下:

```

protected void do_modifyButton_actionPerformed(ActionEvent e) {
    String username = usernameTextField.getText().trim();
    if (username.isEmpty()) {
        JOptionPane.showMessageDialog(this, "姓名不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String title = titleTextField.getText().trim();
    if (title.isEmpty()) {
        JOptionPane.showMessageDialog(this, "主题不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String content = contentTextArea.getText().trim();
    if (content.isEmpty()) {
        JOptionPane.showMessageDialog(this, "内容不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String memotype = memotypeTextField.getText().trim();
    if (memotype.isEmpty()) {
        JOptionPane.showMessageDialog(this, "类型不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String memotime = memotimeTextField.getText().trim();
    if (memotime.isEmpty()) {
        JOptionPane.showMessageDialog(this, "时间不能为空!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (!ValidationUtil.validateTimeFormat(memotime)) {
        JOptionPane.showMessageDialog(this, "日期格式为 2010-12-22 样式!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    MemoBean memo = results.get(index);                // 创建 MemoBean
    memo.setUsername(username);
}

```







```
memo.setTitle(title);
memo.setContent(content);
memo.setMemotype(memotype);
memo.setMemotime(memotime);
int result = JdbcHelper.update(memo);           // 保存备忘录信息
if (result >= 0) {
    JOptionPane.showMessageDialog(this, "备忘录修改成功!");
    return;
} else {
    JOptionPane.showMessageDialog(this, "备忘录修改失败!");
    return;
}
```

## 1.6 查询备忘录

### 1.6.1 功能概述



图 1.12 查询备忘录窗体

为了方便用户查询以前保存的备忘录，本模块提供了查询功能。用户可以使用姓名、类型、时间、主题和内容中任意一项或几项进行模糊查询。单击主窗体“备忘录管理”/“查询备忘录”菜单项，就可以打开“查询备忘录”窗体，如图 1.12 所示。

### 1.6.2 设置文本框和文本区控件

在图 1.12 中，定义了很多文本框和文本区控件用于接收用户输入的信息，并且对不同的面板使用了不同的布局管理器。为了方便用户，使用边框将信息进行分类，其关键代码如下：

```
JPanel othersPanel = new JPanel();           // 创建面板
contentPane.add(othersPanel, BorderLayout.NORTH); // 应用面板
othersPanel.setLayout(new GridLayout(2, 1, 5, 5)); // 为面板设置网格布局

JPanel nttPanel = new JPanel();               // 创建面板
// 设置面板的边框
nttPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_nttPanel = (FlowLayout) nttPanel.getLayout(); // 获得面板布局
fl_nttPanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(nttPanel);                   // 应用面板

JLabel usernameLabel = new JLabel("姓名: "); // 创建标签
usernameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameLabel);                 // 应用标签

usernameTextField = new JTextField();         // 创建文本域
```







```

usernameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameTextField); // 应用文本域
usernameTextField.setColumns(6); // 设置文本域宽度

JLabel memotypeLabel = new JLabel("类型: "); // 创建标签
memotypeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeLabel); // 应用标签

memotypeTextField = new JTextField(); // 创建文本域
memotypeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeTextField); // 应用文本域
memotypeTextField.setColumns(6); // 设置文本域宽度

JLabel memotimeLabel = new JLabel("时间: "); // 创建标签
memotimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeLabel); // 应用标签

memotimeTextField = new JTextField(); // 创建文本域
memotimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeTextField); // 应用文本域
memotimeTextField.setColumns(6); // 设置文本域宽度

JPanel titlePanel = new JPanel(); // 创建面板
// 设置面板的边框
titlePanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_titlePanel = (FlowLayout) titlePanel.getLayout();
// 获得面板布局
fl_titlePanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(titlePanel); // 应用面板

JLabel titleLabel = new JLabel("主题: "); // 创建标签
titleLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleLabel); // 应用标签

titleTextField = new JTextField(); // 创建文本域
titleTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleTextField); // 应用文本域
titleTextField.setColumns(28); // 设置文本域宽度

JPanel contentPanel = new JPanel(); // 创建面板
// 设置面板的边框
contentPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 获得面板布局
FlowLayout fl_contentPanel = (FlowLayout) contentPanel.getLayout();
fl_contentPanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
contentPanel.add(contentPanel, BorderLayout.CENTER); // 应用面板

JLabel contentLabel = new JLabel("内容: "); // 创建标签
contentLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
contentPanel.add(contentLabel); // 应用标签

contentTextArea = new JTextArea(); // 创建文本区
contentTextArea.setColumns(28); // 设置文本区列数
contentTextArea.setLineWrap(true); // 设置文本区自动换行
contentTextArea.setRows(5); // 设置文本区行数
contentTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
// 使用文本区创建滚动窗格
JScrollPane contentScrollPane = new JScrollPane(contentTextArea);
contentPanel.add(contentScrollPane); // 应用滚动窗格

```



### 1.6.3 添加工具按钮

在窗体中使用了“查询”、“清空”和“返回”3个按钮，并将它们放置在一个面板中，





其关键的代码如下:

```
JPanel buttonPanel = new JPanel(); // 创建面板
// 设置面板的边框
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用面板

JButton queryButton = new JButton("查询"); // 创建“查询”按钮
// 省略事件处理相关代码
queryButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(queryButton); // 应用按钮

JButton clearButton = new JButton("清空"); // 创建“清空”按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton); // 应用按钮
```

### 1.6.4 查询备忘录信息

通过监听“查询”按钮单击事件,完成对用户输入条件的查询功能,这里要求用户至少输入一项查询条件。如果查询到结果,将在一个新窗体中进行显示,如图 1.13 所示;否则提示用户结果并不存在,如图 1.14 所示。

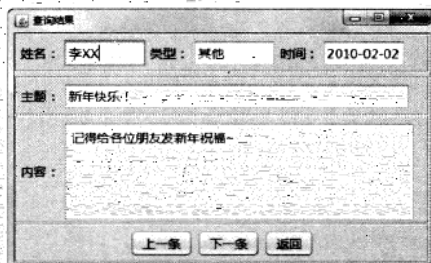


图 1.13 显示查询结果窗体



图 1.14 没有满足条件的记录

“查询”按钮事件监听器关键代码如下:

```
queryButton.addActionListener(new ActionListener() { // 监听按钮事件
    @Override
    public void actionPerformed(ActionEvent e) {
        do_queryButton_actionPerformed(e);
    }
});
```

do\_queryButton\_actionPerformed()方法是 IDE 工具生产的方法,其关键代码如下:

```
protected void do_queryButton_actionPerformed(ActionEvent e) {
    String username = usernameTextField.getText().trim();
    String title = titleTextField.getText().trim();
    String content = contentTextArea.getText().trim();
    String memotype = memotypeTextField.getText().trim();
    String memotime = memotimeTextField.getText().trim();
    if (username.isEmpty() && title.isEmpty() && content.isEmpty()
        && memotype.isEmpty() && memotime.isEmpty()) {
```

```

JOptionPane.showMessageDialog(this, "查询条件不能为空!", "",
JOptionPane.WARNING_MESSAGE);
return;
}
if ((!memotime.isEmpty()) &&
(!ValidationUtil.validateTimeFormat(memotime))) {
JOptionPane.showMessageDialog(this, "日期格式为 2010-12-22 样式!", "",
JOptionPane.WARNING_MESSAGE);
return;
}
MemoBean memo = new MemoBean(); // 创建 MemoBean
memo.setUsername(username);
memo.setTitle(title);
memo.setContent(content);
memo.setMemotype(memotype);
memo.setMemotime(memotime);
final List<MemoBean> results = JdbcHelper.query(memo); // 保存备忘录信息
if (results.size() > 0) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
try {
MemoQueryResultsFrame frame = new
MemoQueryResultsFrame(results);
frame.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
}
}
});
dispose();
} else {
JOptionPane.showMessageDialog(this, "没有符合条件的记录!");
return;
}
}
}

```

## 1.7 显示查询结果

### 1.7.1 功能概述

在完成查询后，将获得的查询结果显示在一个新窗体中，如图 1.15 所示。单击“上一条”按钮和“下一条”按钮可以显示其他的满足条件的结果。

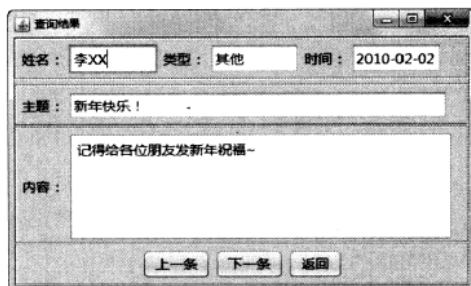


图 1.15 “查询结果”窗体





## 1.7.2 设置文本框和文本区控件

在图 1.15 中, 定义了很多文本框和文本区控件用于显示用户查询的结果, 并且对不同的面板使用了不同的布局管理器。为了方便用户, 使用边框将信息进行分类, 其关键代码如下:

```
JPanel othersPanel = new JPanel(); // 创建面板
contentPane.add(othersPanel, BorderLayout.NORTH); // 应用面板
othersPanel.setLayout(new GridLayout(2, 1, 5, 5)); // 为面板设置网格布局

JPanel nttPanel = new JPanel(); // 创建面板
// 设置面板的边框
nttPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_nttPanel = (FlowLayout) nttPanel.getLayout(); // 获得面板布局
fl_nttPanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(nttPanel); // 应用面板

JLabel usernameLabel = new JLabel("姓名: "); // 创建标签
usernameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameLabel); // 应用标签

usernameTextField = new JTextField(); // 创建文本域
usernameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameTextField); // 应用文本域
usernameTextField.setColumns(6); // 设置文本域宽度

JLabel memotypeLabel = new JLabel("类型: "); // 创建标签
memotypeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeLabel); // 应用标签

memotypeTextField = new JTextField(); // 创建文本域
memotypeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeTextField); // 应用文本域
memotypeTextField.setColumns(6); // 设置文本域宽度

JLabel memotimeLabel = new JLabel("时间: "); // 创建标签
memotimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeLabel); // 应用标签

memotimeTextField = new JTextField(); // 创建文本域
memotimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeTextField); // 应用文本域
memotimeTextField.setColumns(6); // 设置文本域宽度

JPanel titlePanel = new JPanel(); // 创建面板
// 设置面板的边框
titlePanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_titlePanel = (FlowLayout) titlePanel.getLayout(); // 获得面板布局
fl_titlePanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(titlePanel); // 应用面板

JLabel titleLabel = new JLabel("主题: "); // 创建标签
titleLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleLabel); // 应用标签

titleTextField = new JTextField(); // 创建文本域
titleTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleTextField); // 应用文本域
titleTextField.setColumns(28); // 设置文本域宽度
JPanel contentPanel = new JPanel(); // 创建面板
contentPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 设置面板的边框
```







```
FlowLayout fl_contentPanel = (FlowLayout) contentPanel.getLayout();
// 获得面板布局
fl_contentPanel.setAlignment(FlowLayout.LEFT);           // 面板中的控件采用左对齐
contentPane.add(contentPanel, BorderLayout.CENTER);       // 应用面板
JLabel contentLabel = new JLabel("内容:");               // 创建标签
contentLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
contentPanel.add(contentLabel);                           // 应用标签
contentTextArea = new JTextArea();                       // 创建文本区
contentTextPane.add(contentTextArea, BorderLayout.SOUTH); // 设置文本区列数
contentTextArea.setColumns(28);                          // 设置文本区自动换行
contentTextArea.setLineWrap(true);                      // 设置文本区行数
contentTextArea.setRows(5);                             // 设置字体
contentTextPane.add(contentTextArea, BorderLayout.SOUTH); // 设置字体
// 使用文本区创建滚动窗格
JScrollPane contentScrollPane = new JScrollPane(contentTextArea);
contentPanel.add(contentScrollPane);                     // 应用滚动窗格
```

### 1.7.3 添加工具按钮

在窗体中使用了“上一条”、“下一条”和“返回”3个按钮，并将它们放置在一个面板中，其关键的代码如下：

```
JPanel buttonPanel = new JPanel();           // 创建面板
// 设置面板的边框
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用面板
JButton previousButton = new JButton("上一条"); // 创建“上一条”按钮
// 省略事件处理相关代码
previousButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(previousButton); // 应用按钮
JButton nextButton = new JButton("下一条"); // 创建“下一条”按钮
// 省略事件处理相关代码
nextButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(nextButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton); // 应用按钮
```

### 1.7.4 填充备忘录信息

在获得查询结果后，使用自定义的 updateContent() 方法根据索引值添加备忘录信息到窗体，该方法的关键代码如下：

```
private void updateContent(int index) {
    MemoBean memo = results.get(index); // 获得第 index 条备忘录信息
    usernameTextField.setText(memo.getUsername());
    titleTextField.setText(memo.getTitle());
    contentTextArea.setText(memo.getContent());
    memotypeTextField.setText(memo.getMemotype());
    memotimeTextField.setText(memo.getMemotime());
}
```





### 1.7.5 查看上一条查询结果

单击“上一条”按钮，将显示上一条查询结果。在 MemoQueryResultsFrame 类中，定义的事件监听器关键代码如下：

```
previousButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        do_previousButton_actionPerformed(e);  
    }  
});
```

在事件监听器中，调用了 do\_previousButton\_actionPerformed() 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是第一条记录，则提示用户，其关键代码如下：

```
protected void do_previousButton_actionPerformed(ActionEvent e) {  
    if (index <= 0) {  
        JOptionPane.showMessageDialog(this, "已经是第一条记录!", "",  
JOptionPane.WARNING_MESSAGE);  
        return;  
    }  
    updateContent(--index);           //显示上一条信息  
}
```



说明 index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

### 1.7.6 查看下一条查询结果

单击“下一条”按钮，将显示下一条查询结果。在 MemoQueryResultsFrame 类中，定义的事件监听器关键代码如下：

```
nextButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        do_nextButton_actionPerformed(e);  
    }  
});
```

在事件监听器中，调用了 do\_nextButton\_actionPerformed() 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是最后一条记录，则提示用户，其关键代码如下：

```
protected void do_nextButton_actionPerformed(ActionEvent e) {  
    if (index >= (results.size() - 1)) {  
        JOptionPane.showMessageDialog(this, "已经是最后一条记录!", "",  
JOptionPane.WARNING_MESSAGE);  
        return;  
    }  
    updateContent(++index);           //显示下一条信息  
}
```



30







说明 index 是指 List 接口中元素的索引, 因此是从 0 开始计数的。

## 1.8 删除备忘录

### 1.8.1 功能概述

对于已经不需要的备忘录信息, 用户可以将其删除。单击主窗体“备忘录管理”/“删除备忘录”菜单项, 就可以打开删除备忘录窗体。通过单击“上一条”、“下一条”按钮, 可以选择要删除的信息, 单击“删除”按钮完成删除功能, 如图 1.16 所示。

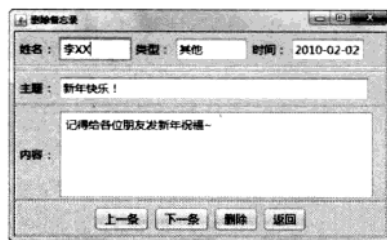


图 1.16 “删除备忘录”窗体

### 1.8.2 设置文本框和文本区控件

在图 1.16 中, 定义了很多文本框和文本区控件用于显示备忘录信息, 并且对不同的面板使用了不同的布局管理器。为了方便用户, 使用边框将信息进行分类, 其关键代码如下:

```

JPanel othersPanel = new JPanel();           // 创建面板
contentPane.add(othersPanel, BorderLayout.NORTH); // 应用面板
othersPanel.setLayout(new GridLayout(2, 1, 5, 5)); // 为面板设置网格布局

JPanel nttPanel = new JPanel();               // 创建面板
// 设置面板的边框
nttPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_nttPanel = (FlowLayout) nttPanel.getLayout(); // 获得面板布局
fl_nttPanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(nttPanel); // 应用面板

JLabel usernameLabel = new JLabel("姓名: "); // 创建标签
usernameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameLabel); // 应用标签

usernameTextField = new JTextField(); // 创建文本域
usernameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(usernameTextField); // 应用文本域
usernameTextField.setColumns(6); // 设置文本域宽度

JLabel memotypeLabel = new JLabel("类型: "); // 创建标签
memotypeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeLabel); // 应用标签

memotypeTextField = new JTextField(); // 创建文本域
memotypeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotypeTextField); // 应用文本域
memotypeTextField.setColumns(6); // 设置文本域宽度

JLabel memotimeLabel = new JLabel("时间: "); // 创建标签
memotimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeLabel); // 应用标签

memotimeTextField = new JTextField(); // 创建文本域

```





```

memotimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
nttPanel.add(memotimeTextField); // 应用文本域
memotimeTextField.setColumns(6); // 设置文本域宽度

JPanel titlePanel = new JPanel(); // 创建面板
// 设置面板的边框
titlePanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
FlowLayout fl_titlePanel = (FlowLayout) titlePanel.getLayout(); // 获得面板布局
fl_titlePanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
othersPanel.add(titlePanel); // 应用面板

JLabel titleLabel = new JLabel("主题:"); // 创建标签
titleLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleLabel); // 应用标签

titleTextField = new JTextField(); // 创建文本域
titleTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
titlePanel.add(titleTextField); // 应用文本域
titleTextField.setColumns(28); // 设置文本域宽度

JPanel contentPanel = new JPanel(); // 创建面板
contentPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 设置面板的边框
FlowLayout fl_contentPanel = (FlowLayout) contentPanel.getLayout();
// 获得面板布局
fl_contentPanel.setAlignment(FlowLayout.LEFT); // 面板中的控件采用左对齐
contentPanel.add(contentPanel, BorderLayout.CENTER); // 应用面板

JLabel contentLabel = new JLabel("内容:"); // 创建标签
contentLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
contentPanel.add(contentLabel); // 应用标签

contentTextArea = new JTextArea(); // 创建文本区
contentTextArea.setColumns(28); // 设置文本区列数
contentTextArea.setLineWrap(true); // 设置文本区自动换行
contentTextArea.setRows(5); // 设置文本区行数
contentTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
// 使用文本区创建滚动窗格
JScrollPane contentScrollPane = new JScrollPane(contentTextArea);
contentPanel.add(contentScrollPane); // 应用滚动窗格

```

### 1.8.3 添加工具按钮

在窗体中使用了“上一条”、“下一条”、“删除”和“返回”4个按钮，并将它们放置在一个面板中，其关键的代码如下：

```

JPanel buttonPanel = new JPanel(); // 创建面板
// 设置面板的边框
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
contentPanel.add(buttonPanel, BorderLayout.SOUTH); // 应用面板

JButton previousButton = new JButton("上一条"); // 创建“上一条”按钮
// 省略事件处理相关代码
previousButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(previousButton); // 应用按钮

JButton nextButton = new JButton("下一条"); // 创建“下一条”按钮
// 省略事件处理相关代码
nextButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(nextButton); // 应用按钮

JButton deleteButton = new JButton("删除"); // 创建“删除”按钮
// 省略事件处理相关代码
deleteButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(deleteButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton); // 应用按钮

```







### 1.8.4 填充备忘录信息

在获得表中保存的全部备忘录信息后,使用自定义的 `updateContent()`方法根据索引值添加备忘录信息到窗体,该方法的关键代码如下:

```
private void updateContent(int index) {
    MemoBean memo = results.get(index);           // 获得第 index 条备忘录信息
    usernameTextField.setText(memo.getUsername());
    titleTextField.setText(memo.getTitle());
    contentTextArea.setText(memo.getContent());
    memotypeTextField.setText(memo.getMemotype());
    memotimeTextField.setText(memo.getMemotime());
}
```

### 1.8.5 删除上一条备忘录信息

单击“上一条”按钮,将显示上一条备忘录信息,用户可以将其删除。在 `MemoDeletionFrame` 类中,定义的事件监听器关键代码如下:

```
previousButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_previousButton_actionPerformed(e);
    }
});
```

在事件监听器中,调用了 `do_previousButton_actionPerformed()`方法,它是由 IDE 工具自动生成的,这里需要对当前位置进行校验。如果已经是第一条记录,则提示用户,其关键代码如下:

```
protected void do_previousButton_actionPerformed(ActionEvent e) {
    if (index <= 0) {
        JOptionPane.showMessageDialog(this, "已经是第一条记录!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    updateContent(--index);           // 显示上一条信息
}
```



**说明** index 是指 List 接口中元素的索引,因此是从 0 开始计数的。

### 1.8.6 删除下一条备忘录信息

单击“下一条”按钮,将显示下一条备忘录信息,用户可以将其删除。在 `MemoDeletionFrame` 类中,定义的事件监听器关键代码如下:

```
nextButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_nextButton_actionPerformed(e);
    }
});
```

在事件监听器中,调用了 `do_nextButton_actionPerformed()`方法,它是由 IDE 工具自





动生成的，这里需要对当前位置进行校验。如果已经是最后一条记录，则提示用户，其关键代码如下：

```
protected void do_nextButton_actionPerformed(ActionEvent e) {
    if (index >= (results.size() - 1)) {
        JOptionPane.showMessageDialog(this, "已经是最后一条记录!", "",
        JOptionPane.WARNING_MESSAGE);
        return;
    }
    updateContent(++index); //显示下一条信息
}
```



index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

### 1.8.7 删除备忘录信息

通过监听“删除”按钮单击事件，完成对当前显示信息的删除功能。“删除”按钮事件监听器关键代码如下：

```
deleteButton.addActionListener(new ActionListener() { // 监听按钮事件
    @Override
    public void actionPerformed(ActionEvent e) {
        do_deleteButton_actionPerformed(e);
    }
});
```

do\_deleteButton\_actionPerformed()方法是 IDE 工具生产的方法，其关键代码如下：

```
protected void do_deleteButton_actionPerformed(ActionEvent e) {
    JdbcHelper.delete(results.get(index));
    results.remove(index); // 删除当前记录
    if (index != 0) {
        index--;
    }
    if (results.size() == 0) { // 如果表格中已经没有数据，则提示用户
        usernameTextField.setText("");
        titleTextField.setText("");
        contentTextArea.setText("");
        memotypeTextField.setText("");
        memotimeTextField.setText("");
        JOptionPane.showMessageDialog(this, "备忘录为空!", "",
        JOptionPane.WARNING_MESSAGE);
        return;
    } else { // 否则使用前一条记录填充控件
        updateContent(index);
        JOptionPane.showMessageDialog(this, "备忘录删除成功!", "",
        JOptionPane.WARNING_MESSAGE);
        return;
    }
}
```





# 第 2 章

---

## 学生成绩管理模块

( Swing+MySQL 实现 )

随着教育的不断普及，各个学校的学生人数也越来越多。传统的管理方式并不能适应时代的发展，为了提高管理效率，减少学校开支，使用软件管理学生已成为必然。本章将开发一个学生成绩管理模块，通过本章的学习，可以掌握以下要点：

- » Swing 控件的使用
- » 绘制艺术文字
- » 使用 JDBC 操作数据库
- » 使用正则表达式校验日期
- » 调用系统小工具



## 2.1 学生成绩管理模块概述

### 2.1.1 模块概述

学生成绩管理工作一直被视为校园管理中的一个瓶颈,积极寻求适应时代要求的学生成绩管理模式已经成为当前校园管理工作的当务之急。传统的人力管理模式既浪费人力,同时效果不够明显。本章将开发一个学生成绩管理模块,它包括了学生成绩的增、删、改、查功能,同时也可以调用 Windows 自带的小工具。读者可以在本模块的基础上开发学生成绩管理系统。

### 2.1.2 功能结构

学生成绩管理模块主要包括增加成绩单、修改成绩单、查询成绩单和删除成绩单功能。此外,还可以打开 Windows 系统的记事本和计算器小工具,其功能结构如图 2.1 所示。

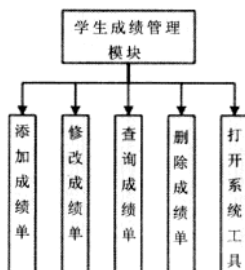


图 2.1 学生成绩管理模块功能结构

### 2.1.3 程序预览

学生成绩模块由多个窗体组成,主窗体的运行效果如图 2.2 所示,主要功能是调用执行本模块的所有功能。

“增加成绩单”窗体的运行效果如图 2.3 所示,主要功能是完成成绩单的增加。除了备注,所有内容都是必须填写的,其中,学生班级的格式要求是例如“0123”的样式。第一个数字表示初中还是高中,第二个数字表示年级,后面两个数字表示具体的班级。第一位使用 0 表示初中,1 表示高中。第二位使用 1、2 和 3 表示年级。考试成绩的格式要求是例如“90”的样式。考试时间的格式要求是例如“2010-12-22”的样式,否则将不能增加成绩单。

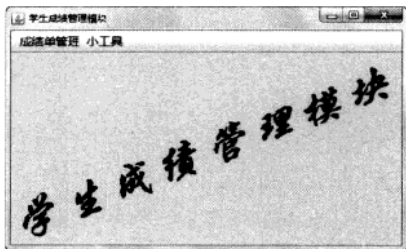


图 2.2 学生成绩管理模块主窗体

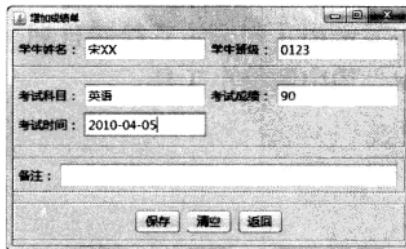


图 2.3 “增加成绩单”窗体





“修改成绩单”窗体主要功能是完成对已经增加的成绩单的修改。修改时将首先查询出数据库中保存的所有成绩单，如图 2.4 所示。用户要选择一行记录，然后单击“修改”按钮进行修改，如图 2.5 所示。

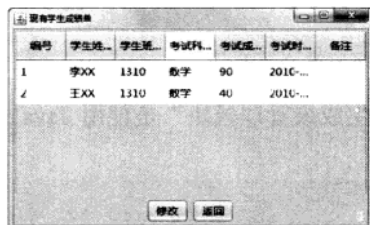


图 2.4 “现有学生成绩单”窗体

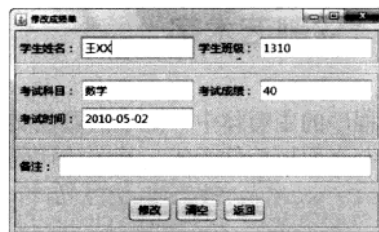


图 2.5 “修改成绩单”窗体

“查询成绩单”窗体的运行效果如图 2.6 所示，主要功能是对已经保存的成绩单进行查询。

查询的条件是任意的，而且支持模糊查询。如果没有找到符合条件的记录，则进行提示，如图 2.7 所示。

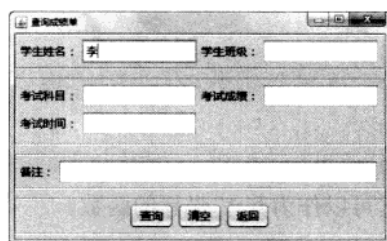


图 2.6 “查询成绩单”窗体

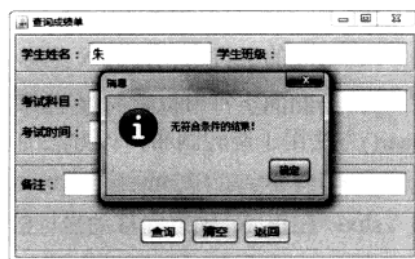


图 2.7 查询失败时的提示窗体

否则将查询结果显示在表格中，如图 2.8 所示。

“删除学生成绩单”窗体如图 2.9 所示，它用于删除当前窗体表格中选中的学生成绩单记录。

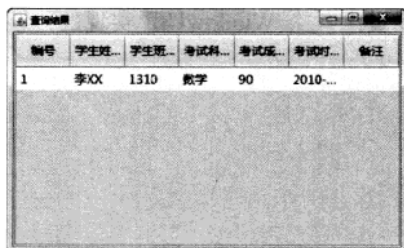


图 2.8 查询成功时的窗体

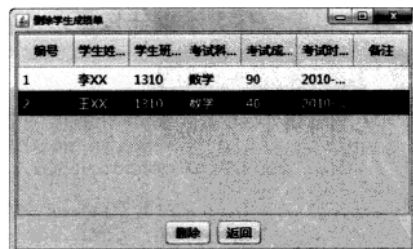


图 2.9 “删除学生成绩单”窗体





## 2.2 关键技术

### 2.2.1 绘制艺术字

在程序的主窗体中，显示的彩色文本信息“学生成绩管理模块”是使用 Java 的绘图技术绘制在面板控件上的，其关键代码如下：

```
public void paint(Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;           // 获得 Graphics2D 对象  
    g2.setFont(new Font("华文行楷", Font.BOLD, 56)); // 设置字体  
    g2.shear(0.1, -0.4);                       // 倾斜画布  
    g2.drawString("学生成绩管理模块", -15, 230); // 绘制文本  
}
```

这里主要使用 Graphics2D 类中的 drawString 方法来完成字符串的绘制，该方法的声明如下：

```
public abstract void drawString(String str,int x,int y)
```

- str: 需要绘制的字符串。
- x: 绘制的字符串位置的 X 坐标。
- y: 绘制的字符串位置的 Y 坐标。

shear()方法用于旋转画布，该方法的声明如下：

```
public abstract void shear(double shx,double shy)
```

- shx: 在正 X 轴方向移动坐标的乘数，它可以作为其 Y 坐标的函数。
- shy: 在正 Y 轴方向移动坐标的乘数，它可以作为其 X 坐标的函数。

### 2.2.2 窗体居中显示

为了让使用者更加舒适，可以设置窗体显示位置为居中显示，这就需要获得用户显示器的大小并根据窗体大小来计算显示位置，这是通过工具类 WindowUtil 实现的，该类的代码如下：

```
public class WindowUtil {  
    // 将窗体大小设置成 500×309  
    public static Dimension getSize() {  
        return new Dimension(500, 309);  
    }  
    // 计算窗体居中显示时左上角坐标  
    public static Point getLocation() {  
        Toolkit toolKit = Toolkit.getDefaultToolkit(); // 获得 Toolkit 实例  
        Dimension screenSize = toolKit.getScreenSize(); // 获得显示器大小  
        if ((screenSize.width < getSize().width) || (screenSize.height <   
            getSize().height)) {  
            JOptionPane.showMessageDialog(null, "显示器分辨率至少为 500×309", "",  
                JOptionPane.WARNING_MESSAGE);  
            System.exit(0); // 终止程序  
        }  
        int x = (screenSize.width - getSize().width) / 2; // 计算左上角横坐标  
    }  
}
```







```

        int y = (screenSize.height - getSize().height) / 2; // 计算左上角纵坐标
        return new Point(x, y);
    }
}

```

首先定义 `getSize()` 方法, 它返回窗体的大小, 这里将窗体的宽度设置为 500 像素, 高度设置为 309 像素。Dimension 类用于封装控件的宽度和高度, 使用该类有利于以面向对象的方式编写代码。

接着定义 `getLocation()` 方法, 它返回窗体左上角的坐标。这里首先获得 Toolkit 类的对象, 由于该类是抽象类, 不能直接使用 `new` 操作符实例化, 因此使用该类定义的 `getDefaultToolkit()` 方法获得实例。该方法的声明如下:

```
public static Toolkit getDefaultToolkit()
```

Toolkit 类中的 `getScreenSize()` 方法可以获得用户显示器的分辨率, 该方法的声明如下:

```
public abstract Dimension getScreenSize() throws HeadlessException
```

如果用户的分辨率小于  $500 \times 309$  像素则弹出提示对话框并终止程序。

### 2.2.3 使用 JavaBean 封装信息

在本模块中, 用户需要填写成绩单, 这涉及了大量的信息, 例如学生姓名、学生班级、考试科目等, 通常定义一个类来保存这些信息, 将它们设置成私有字段, 并提供对应的 `get` 和 `set` 方法, 这样就构成了一个 JavaBean。GradeBean 的代码如下:

```

public class GradeBean {
    private int id;                // 成绩编号
    private String studentName;    // 学生姓名
    private String studentClass;   // 学生班级
    private String testSubject;    // 考试科目
    private String score;          // 考试成绩
    private String testTime;       // 考试时间
    private String remark;         // 备注
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    public String getStudentClass() {
        return studentClass;
    }
    public void setStudentClass(String studentClass) {
        this.studentClass = studentClass;
    }
    public String getTestSubject() {
        return testSubject;
    }
    public void setTestSubject(String testSubject) {
        this.testSubject = testSubject;
    }
    public String getScore() {

```





```
        return score;
    }
    public void setScore(String score) {
        this.score = score;
    }
    public String getTestTime() {
        return testTime;
    }
    public void setTestTime(String testTime) {
        this.testTime = testTime;
    }
    public String getRemark() {
        return remark;
    }
    public void setRemark(String remark) {
        this.remark = remark;
    }
}
```

## 2.2.4 获得 MySQL 数据库连接

本程序的后台使用 MySQL 数据库来保存成绩单信息，在使用 JDBC 操作数据库时，需要先获得数据库连接对象，这是使用 JdbcHelper 类的 getConnection() 方法实现的，该方法的代码如下：

```
private static Connection getConnection() {
    Connection conn = null;
    try {
        Class.forName(DRIVER);           // 加载数据库驱动

        // 获得数据库连接
        conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        return conn;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

方法中用到的 DRIVER、URL、USERNAME 和 PASSWORD 是字符串常量，定义在 JdbcConfig 接口中，该接口的代码如下：

```
public interface JdbcConfig {
    String DRIVER = "com.mysql.jdbc.Driver";
    String URL = "jdbc:mysql://localhost:3306/db_studentgrademodule";
    String USERNAME = "root";
    String PASSWORD = "111";
}
```



在使用 JDBC 操作数据库时需要为项目添加数据库驱动文件。





## 2.2.5 批量处理数据库操作

在处理大量数据时,使用批处理操作能够大幅度提高效率,这里使用该技术完成成绩单的保存和修改操作。

### 1. 保存成绩单信息

使用 JdbcHelper 类的 save() 方法来保存成绩单信息,其关键代码如下:

```
public static int save(GradeBean grade) {
    String sql = "insert into tb_transcript (studentName, studentClass,
        testSubject, score, testTime, remark)
        values (?, ?, ?, ?, ?, ?)";
    Connection conn = getConnection();
    PreparedStatement ps = null;
    try {
        ps = conn.prepareStatement(sql);
        ps.setString(1, grade.getStudentName());
        ps.setString(2, grade.getStudentClass());
        ps.setString(3, grade.getTestSubject());
        ps.setString(4, grade.getScore());
        ps.setString(5, grade.getTestTime());
        ps.setString(6, grade.getRemark());
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return -1;
}
```

使用 PreparedStatement 接口可以完成批处理操作。批处理时使用问号来表示需要添加的变量,然后使用该接口中与变量类型对应的 set() 方法来添加变量。例如 setString() 表示设置字符串的值,这个方法的第一个参数表示变量的位置,第二个参数表示变量的值。最后调用 executeUpdate() 方法来完成批量保存。

### 2. 修改成绩单

修改成绩单使用的是 JdbcHelper 类中的 update() 方法,其关键代码如下:

```
public static int update(GradeBean grade) {
    String sql = "update tb_transcript set studentName = ?, studentClass = ?,
        testSubject = ?,
        score = ?, testTime = ?, remark = ? where id = ?";
    Connection conn = getConnection();
    PreparedStatement ps = null;
    try {
```







```
        ps = conn.prepareStatement(sql);
        ps.setString(1, grade.getStudentName());
        ps.setString(2, grade.getStudentClass());
        ps.setString(3, grade.getTestSubject());
        ps.setString(4, grade.getScore());
        ps.setString(5, grade.getTestTime());
        ps.setString(6, grade.getRemark());
        ps.setInt(7, grade.getId());
        return ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return -1;
}
```

## 2.2.6 使用 List 保存查询结果

在 JDBC 中，查询操作的返回值是 `ResultSet` 类型的对象，它使用起来并不方便，因此通常将结果保存到 `List` 中。

### 1. 保存全部成绩单信息

本模块使用 `JdbcHelper` 类的 `queryAll()` 方法来保存表格中全部的数据，其关键代码如下：

```
public static List<GradeBean> queryAll() {
    String sql = "select * from tb_transcript;";
    List<GradeBean> results = new ArrayList<GradeBean>();
    Connection conn = getConnection();
    Statement stat = null;
    ResultSet rs = null;
    try {
        stat = conn.createStatement();
        rs = stat.executeQuery(sql);
        while (rs.next()) {
            GradeBean grade = new GradeBean();
            grade.setId(rs.getInt("id"));
            grade.setStudentName(rs.getString("studentName"));
            grade.setStudentClass(rs.getString("studentClass"));
            grade.setTestSubject(rs.getString("testSubject"));
            grade.setScore(rs.getString("score"));
            grade.setTestTime(rs.getString("testTime"));
            grade.setRemark(rs.getString("remark"));
            results.add(grade);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        // 释放资源
    }
}
```







```

        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return results;
}

```

## 2. 保存部分成绩单信息

本模块使用 JdbcHelper 类的 query () 方法来保存满足查询条件的全部数据, 这里将查询条件封装到 GradeBean 对象中, 其关键代码如下:

```

public static List<GradeBean> query(GradeBean grade) {
    String studentName = grade.getStudentName();
    String studentClass = grade.getStudentClass();
    String testSubject = grade.getTestSubject();
    String score = grade.getScore();
    String testTime = grade.getTestTime();
    String remark = grade.getRemark();

    StringBuilder sql = new StringBuilder("select * from tb_transcript where l=1 ");
    if (!studentName.isEmpty()) {
        sql.append(" and studentName like '%" + studentName + "%' ");
    }
    if (!studentClass.isEmpty()) {
        sql.append(" and studentClass like '%" + studentClass + "%' ");
    }
    if (!testSubject.isEmpty()) {
        sql.append(" and testSubject like '%" + testSubject + "%' ");
    }
    if (!score.isEmpty()) {
        sql.append(" and score like '%" + score + "%' ");
    }
    if (!testTime.isEmpty()) {
        sql.append(" and testTime like '%" + testTime + "%' ");
    }
    if (!remark.isEmpty()) {
        sql.append(" and remark like '%" + remark + "%' ");
    }
    sql.append(";");
    List<GradeBean> results = new ArrayList<GradeBean>();
    Connection conn = getConnection();
    Statement stat = null;
    ResultSet rs = null;
    try {
        stat = conn.createStatement();
        rs = stat.executeQuery(sql.toString());
        while (rs.next()) {

```







```
GradeBean tempGrade = new GradeBean();
tempGrade.setId(rs.getInt("id"));
tempGrade.setStudentName(rs.getString("studentName"));
tempGrade.setStudentClass(rs.getString("studentClass"));
tempGrade.setTestSubject(rs.getString("testSubject"));
tempGrade.setScore(rs.getString("score"));
tempGrade.setTestTime(rs.getString("testTime"));
tempGrade.setRemark(rs.getString("remark"));
results.add(tempGrade);
}
} catch (SQLException e) {
    e.printStackTrace();
} finally { // 释放资源
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (stat != null) {
        try {
            stat.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
return results;
}
```

## 2.2.7 使用正则表达式进行校验

在输入时间时，需要用户按照类似“2010-12-22”的样式输入，可以使用正则表达式完成输入格式的校验工作，但是非常复杂，这里使用 Java 代码来辅助校验，其关键代码如下：

```
public static boolean validateTestTimeFormat(String memotime) {
    String regx =
        "([123]\\d{3}-([0][1-9])|([1][0-2]))-([0][1-9])|([12]\\d)|([3][01]))";
    Pattern pattern = Pattern.compile(regx);
    Matcher matcher = pattern.matcher(memotime);
    if (!matcher.matches()) {
        return false;
    } else {
        String[] times = memotime.split("-");
        Integer year = Integer.parseInt(times[0]); // 将输入的时间用-分割
        Integer month = Integer.parseInt(times[1]); // 获得用户输入的年
        Integer day = Integer.parseInt(times[2]); // 获得用户输入的月
        GregorianCalendar calendar = new GregorianCalendar(); // 获得用户输入的天
        if (calendar.isLeapYear(year) && (month == 2) && (day > 29)) {
            // 闰年 2 月天数不能大于 29
            return false;
        } else if ((!calendar.isLeapYear(year)) && (month == 2) && (day > 28)) {
            // 非闰年 2 月天数不能大于 28
        }
    }
}
```







```

        return false;
    } else {
        switch (month) {
            case 4: // 4、6、9、11月都是30天
            ;
            case 6:
            ;
            case 9:
            ;
            case 11:
                if (day > 30) {
                    return false;
                } else {
                    return true;
                }
            default:
                return true;
        }
    }
}
}
}

```

字符串"`([123]\\d{3}-((([0][1-9])|([1][0-2]))-((([0][1-9])|([12]\\d)|([3][01]))))`"是自定义的正则表达式, `[123]`表示 1、2 和 3 三个数字选取一个; `(\\d)`表示 0~9 中任意一个数字; `{3}`表示将前面临近的表达式重复 3 遍, 即 `(\\d){3}` 与 `(\\d)(\\d)(\\d)` 等价。对于年, 其取值范围设置成了 1000~3999。对于月, 合法的格式应该是 01~09 和 10、11、12。使用 `[0][1-9]` 表示 01~09, 使用 `[1][0-2]` 表示 10、11、12。中间的 `|` 表示或。对于日, 合法的格式应该是 01~09、10~29、30、31, 使用 `[0][0-9]` 表示 01~09, 使用 `[12]\\d` 表示 10~29, 使用 `[3][01]` 表示 30 和 31。

但是这里并没有对具体的年份、月份进行校验。例如闰年的二月不能有 30 天、非闰年的二月不能有 29 天。每年的 4、6、9、11 月都是 30 天, 不能有 31 天。使用正则表达式完成这类校验非常复杂, 因此使用 Java 代码来完成。

`Pattern` 类是在 `java.util.regex` 包中定义的正则表达式类, 它的 `compile()` 方法能够解析给定的正则表达式, 该方法的声明如下:

```
public static Pattern compile(String regex)
```

regex: 需要解析的正则表达式。

`Matcher` 类的 `matches()` 方法能够判断给定的字符串是否符合正则表达式, 该方法的声明如下:

```
public boolean matches()
```

## 2.2.8 调用系统工具

在本模块中, 提供了系统工具的调用功能, 用户可以打开 Windows 系统中的记事本和计算器程序, 这是使用 `Runtime` 类完成的, 打开记事本的关键代码如下:

```

Runtime runTime = Runtime.getRuntime(); // 获得 Runtime 类型对象
try {
    runTime.exec("notepad"); // 执行 notepad 命令打开记事本
} catch (IOException e1) {
    e1.printStackTrace();
}

```

`Runtime` 类的 `exec()` 方法可以执行系统命令, 该方法的声明如下:





```
public Process exec(String command) throws IOException
```

command: 本地系统命令。



本程序是在 Windows 系统中开发的，并不适用于 Linux 系统。

## 2.3 主窗体

### 2.3.1 功能概述

学生成绩管理模块由多个窗体组成，主窗体的运行效果如图 2.10 所示，主要功能是调用执行本系统的所有功能。

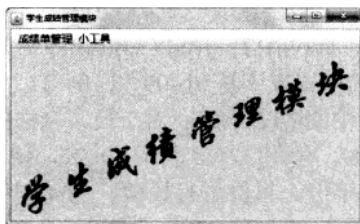


图 2.10 学生成绩管理模块主窗体

### 2.3.2 添加菜单及菜单项

在主窗体中，使用菜单来管理各个功能，下面介绍菜单及菜单项的使用。

在窗体中添加菜单前，需要先添加菜单栏，即 `JMenuBar`，其关键代码如下：

```
JMenuBar menuBar = new JMenuBar();           // 创建菜单条  
setJMenuBar(menuBar);                       // 应用菜单条
```

接着向菜单栏中增加菜单，其关键代码如下：

```
JMenu gradeManagementMenu = new JMenu("成绩单管理"); // 创建菜单  
gradeManagementMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15));  
// 设置字体  
menuBar.add(gradeManagementMenu);           // 应用菜单
```

```
JMenu toolMenu = new JMenu("小工具");          // 创建菜单  
toolMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体  
menuBar.add(toolMenu);                        // 应用菜单
```

这里创建了“成绩单管理”和“小工具”两个菜单。

下面分别为这两个菜单增加菜单项，其关键代码如下：

```
JMenuItem addGradeMenuItem = new JMenuItem("增加成绩单"); // 创建菜单项  
// 省略事件处理相关代码  
addGradeMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体  
gradeManagementMenu.add(addGradeMenuItem); // 应用菜单项
```



46







```

JMenuItem modifyGradeMenuItem = new JMenuItem("修改成绩单"); // 创建菜单项
// 省略事件处理相关代码
modifyGradeMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
gradeManagementMenu.add(modifyGradeMenuItem); // 应用菜单项

JMenuItem queryGradeMenuItem = new JMenuItem("查询成绩单"); // 创建菜单项
// 省略事件处理相关代码
queryGradeMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
gradeManagementMenu.add(queryGradeMenuItem); // 应用菜单项

JMenuItem deleteGradeMenuItem = new JMenuItem("删除成绩单"); // 创建菜单项
// 省略事件处理相关代码
deleteGradeMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
gradeManagementMenu.add(deleteGradeMenuItem); // 应用菜单项

JMenuItem notepadMenuItem = new JMenuItem("记事本"); // 创建菜单项
// 省略事件处理相关代码
notepadMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
toolMenu.add(notepadMenuItem); // 应用菜单项

JMenuItem calculatorMenuItem = new JMenuItem("计算器"); // 创建菜单项
// 省略事件处理相关代码
calculatorMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
toolMenu.add(calculatorMenuItem); // 应用菜单项

```

在上述代码中, 统一将字体设置成微软雅黑, 大小是 15 磅。

为了处理用户单击菜单项事件, 可以为其增加事件监听器, 下面以“增加成绩单”菜单项为例进行讲解。添加事件监听器的关键代码如下:

```

addGradeMenuItem.addActionListener(new ActionListener() { // 监听菜单项事件
    @Override
    public void actionPerformed(ActionEvent e) {
        do_addGradeMenuItem_actionPerformed(e);
    }
});

```

在事件监听器中, 调用了 `do_addGradeMenuItem_actionPerformed()` 方法, 它是由 IDE 工具自动生成的, 用于创建 `GradeAdditionFrame` 并将其设置成可见, 其关键代码如下:

```

protected void do_addGradeMenuItem_actionPerformed(ActionEvent e) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                GradeAdditionFrame frame = new GradeAdditionFrame();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

```

### 2.3.3 绘制窗体中的艺术字

为了美化程序, 这里采用绘制艺术字的方式, 它是通过重写 `JPanel` 控件的 `paint()` 方法完成的, 其关键代码如下:

```

contentPane = new JPanel() {
    private static final long serialVersionUID = 2107432584599709079L;

```





```
@Override
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;           // 获得 Graphics2D 对象
    g2.setFont(new Font("华文行楷", Font.BOLD, 56)); // 设置字体
    g2.shear(0.1, -0.4);                       // 倾斜画布
    g2.drawString("学生成绩管理模块", -15, 230); // 绘制文本
}
};
```

## 2.3.4 设置窗体显示位置和大小

在使用窗体之前，需要为其设置显示位置和大小，因为 Swing 中窗体的默认大小是 0×0。这里使用自定义的 WindowUtil 类，其关键代码如下：

```
setLocation(WindowUtil.getLocation()); // 设置窗体显示位置
setSize(WindowUtil.getSize());        // 设置窗体大小
```



对于其他的窗体也采用类似的方式设置显示位置和大小，下面省略这部分的

讲解。

至此完成了 MainFrame 类的编写，关于该类的详细代码请读者参考源文件。

## 2.4 增加成绩单

### 2.4.1 功能概述

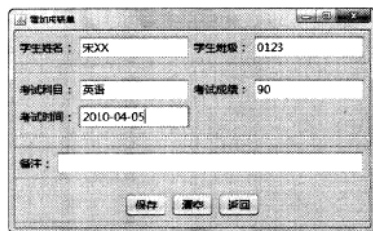


图 2.11 “增加成绩单”窗体

增加成绩单窗体用于增加成绩单，包括学生姓名、考试科目、考试成绩等相关信息。单击主窗体“成绩单管理”/“增加成绩单”菜单项，就可以打开“增加成绩单”窗体，如图 2.11 所示。



### 2.4.2 配置非按钮控件

48



在图 2.11 中，定义了若干文本框控件用于接收学生姓名、学生班级、考试科目等信息，并使用标签来标示各个文本框的作用。为了便于分类，使用边框将相关信息进行了分组，其关键代码如下：

```
JPanel studentInfoPanel = new JPanel(); // 创建面板
studentInfoPanel.setBounds(0, 0, 484, 46); // 设置面板位置
```





```

studentInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null,
null)); // 设置面板边框
FlowLayout flowLayout = (FlowLayout) studentInfoPanel.getLayout();
// 获得面板布局
flowLayout.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(studentInfoPanel); // 应用面板

JLabel studentNameLabel = new JLabel("学生姓名:"); // 创建标签
studentNameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameLabel); // 应用标签

studentNameTextField = new JTextField(); // 创建文本框
studentNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameTextField); // 应用文本框
studentNameTextField.setColumns(10); // 设置文本框显示的列数

JLabel studentClassLabel = new JLabel("学生班级:"); // 创建标签
studentClassLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassLabel); // 应用标签

studentClassTextField = new JTextField(); // 创建文本框
studentClassTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassTextField); // 应用文本框
studentClassTextField.setColumns(10); // 设置文本框显示的列数

testInfoPanel = new JPanel(); // 创建面板
testInfoPanel.setBounds(0, 58, 484, 89); // 设置面板位置
// 设置面板边框
testInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 获得面板布局
FlowLayout flowLayout_1 = (FlowLayout) testInfoPanel.getLayout();
flowLayout_1.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(testInfoPanel); // 应用面板

testSubjectLabel = new JLabel("考试科目:"); // 创建标签
testSubjectLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectLabel); // 应用标签

testSubjectTextField = new JTextField(); // 创建文本框
testSubjectTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectTextField); // 应用文本框
testSubjectTextField.setColumns(10); // 设置文本框显示的列数

scoreLabel = new JLabel("考试成绩:"); // 创建标签
scoreLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreLabel); // 应用标签

scoreTextField = new JTextField(); // 创建文本框
scoreTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreTextField); // 应用文本框
scoreTextField.setColumns(10); // 设置文本框显示的列数

testTimeLabel = new JLabel("考试时间:"); // 创建标签
testTimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeLabel); // 应用标签

testTimeTextField = new JTextField(); // 创建文本框
testTimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeTextField); // 应用文本框
testTimeTextField.setColumns(10); // 设置文本框显示的列数

remarkPanel = new JPanel(); // 创建面板
// 设置面板边框
remarkPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 获得面板布局
FlowLayout flowLayout_2 = (FlowLayout) remarkPanel.getLayout();
flowLayout_2.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列

```







```

remarkPanel.setBounds(0, 159, 484, 46); // 设置面板位置
contentPane.add(remarkPanel); // 应用面板

JLabel remarkLabel = new JLabel("备注:"); // 创建标签
remarkLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkLabel); // 应用标签

remarkTextField = new JTextField(); // 创建文本框
remarkTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkTextField); // 应用文本框
remarkTextField.setColumns(29); // 设置文本框显示的列数

```

### 2.4.3 添加工具按钮

在窗体中定义了“保存”、“清空”和“返回”3个按钮，用于完成保存成绩单、清除已填内容和返回主窗体功能。其关键的代码如下：

```

JPanel buttonPanel = new JPanel(); // 创建面板
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
buttonPanel.setBounds(0, 217, 484, 48); // 设置面板位置
contentPane.add(buttonPanel); // 应用面板

JButton saveButton = new JButton("保存"); // 创建按钮
// 省略事件处理相关代码
saveButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(saveButton); // 应用按钮

JButton clearButton = new JButton("清空"); // 创建按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton); // 应用按钮

```

### 2.4.4 保存成绩单信息

通过监听“保存”按钮单击事件，完成对用户增加的成绩单信息的保存功能。由于成绩单上的信息都非常重要，因此不能为空值（备注信息除外）。“保存”按钮事件监听器关键代码如下：

```

saveButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_saveButton_actionPerformed(e);
    }
});

```



50



在事件监听器中，调用了 `do_saveButton_actionPerformed()` 方法，它是 IDE 工具自动生成的方法，用于对文本域的非空校验及保存用户输入信息，其关键代码如下：

```

protected void do_saveButton_actionPerformed(ActionEvent e) {
    String studentName = studentNameTextField.getText().trim();
    if (studentName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "学生姓名不能为空!", "",

```





```

JOptionPane.WARNING_MESSAGE);
    return;
}
String studentClass = studentClassTextField.getText().trim();
if (studentClass.isEmpty()) {
    JOptionPane.showMessageDialog(this, "学生班级不能为空!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
if (!ValidationUtil.validateStudentClass(studentClass)) {
    JOptionPane.showMessageDialog(this, "学生班级为 0102 样式!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
String testSubject = testSubjectTextField.getText().trim();
if (testSubject.isEmpty()) {
    JOptionPane.showMessageDialog(this, "考试科目不能为空!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
String score = scoreTextField.getText().trim();
if (score.isEmpty()) {
    JOptionPane.showMessageDialog(this, "考试成绩不能为空!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
if (!ValidationUtil.validateScore(score)) {
    JOptionPane.showMessageDialog(this, "考试成绩为 88 样式!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
String testTime = testTimeTextField.getText().trim();
if (testTime.isEmpty()) {
    JOptionPane.showMessageDialog(this, "考试时间不能为空!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
if (!ValidationUtil.validateTestTimeFormat(testTime)) {
    JOptionPane.showMessageDialog(this, "考试时间为 2011-02-23 样式!", "",
JOptionPane.WARNING_MESSAGE);
    return;
}
String remark = remarkTextField.getText().trim();
GradeBean grade = new GradeBean();
grade.setStudentName(studentName);
grade.setStudentClass(studentClass);
grade.setTestSubject(testSubject);
grade.setScore(score);
grade.setTestTime(testTime);
grade.setRemark(remark);
int result = JdbcHelper.save(grade);
if (result >= 0) {
    JOptionPane.showMessageDialog(this, "成绩单增加成功!");
    return;
} else {
    JOptionPane.showMessageDialog(this, "成绩单增加失败!");
    return;
}
}

```



### 2.4.5 清空成绩单信息

在增加完成绩单之后, 可以使用“清空”按钮清除成绩单中的信息, 其事件监听器关



键代码如下：

```
clearButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_clearButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_clearButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，其关键代码如下：

```
protected void do_clearButton_actionPerformed(ActionEvent e) {
    studentNameTextField.setText("");
    studentClassTextField.setText("");
    testSubjectTextField.setText("");
    scoreTextField.setText("");
    testTimeTextField.setText("");
    remarkTextField.setText("");
}
```

### 2.4.6 销毁窗体

在使用完该窗体之后，可以单击“返回”按钮销毁该窗体，其事件监听器关键代码如下：

```
returnButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_returnButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_returnButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，其关键代码如下：

```
protected void do_returnButton_actionPerformed(ActionEvent e) {
    dispose();
}
```

`dispose()` 方法可以释放该窗体及其子控件占用的资源，该方法的声明如下：

```
public void dispose()
```

## 2.5 显示已保存成绩单

### 2.2.1 功能概述



52



在修改成绩单和删除成绩单之前，先显示已经保存的成绩单供用户进行选择。单击主窗体“成绩单管理”/“修改成绩单”菜单项，显示的效果如图 2.12 所示。用户在选择一条记录之后，单击“修改”按钮将打开修改窗体。

单击主窗体“成绩单管理”/“删除成绩单”菜单项，显示的效果如图 2.13 所示。用户在选择一条记录之后，单击“删除”按钮将删除该记录。



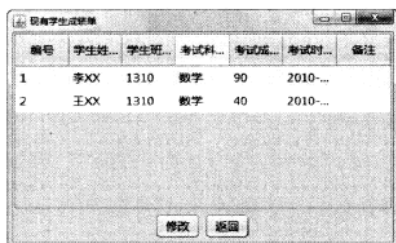


图 2.12 “现有学生成绩单”窗体

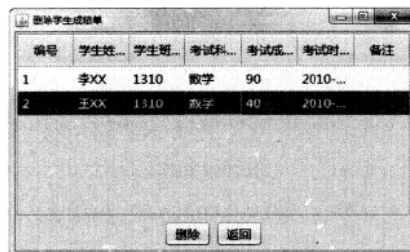


图 2.13 删除学生成绩单窗体

由于这两个窗体主要是按钮的功能不同，因此放在一起进行讲解。

## 2.2.2 为表格控件添加数据

窗体的核心控件是一个表格，由于 Swing 中 JTable 控件表头和表体的高度是固定的，如果修改字体将有部分不能显示，因此首先要修改这个默认值，其关键代码如下：

```
studentGradeTable.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置表体字体
studentGradeTable.setRowHeight(30); // 设置表体高度
JTableHeader header = studentGradeTable.getTableHeader(); // 获得表头对象
header.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置表头字体
header.setPreferredSize(new Dimension(header.getWidth(), 40)); // 设置表头高度
```

为了防止用户同时选择多行记录进行修改，将表格的选择模式设置成单行选择，其代码如下：

```
studentGradeTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

然后使用 DefaultTableModel 来为表格增加数据，数据包括表头和表体，其关键代码如下：

```
DefaultTableModel model = (DefaultTableModel) studentGradeTable.getModel();
// 获得表格模型
model.setRowCount(0); // 清空表格中的数据
model.setColumnIdentifiers(new Object[] { "编号", "学生姓名", "学生班级", "考试科目", "考试成绩", "考试时间", "备注" });
List<GradeBean> results = JdbcHelper.queryAll(); // 获得数据库中表格的全部数据
for (GradeBean grade : results) { // 将数据加载到表格模型中
    model.addRow(new Object[] { grade.getId(), grade.getStudentName(),
        grade.getStudentClass(),
        grade.getTestSubject(),
        grade.getScore(), grade.getTestTime(), grade.getRemark() });
}
studentGradeTable.setModel(model); // 应用表格模型
```

## 2.2.3 “修改”按钮事件监听

“修改”按钮用于处理修改用户选择的表格中的记录，其事件监听器关键代码如下：

```
protected void do_modifyButton_actionPerformed(ActionEvent e) {
    int selectRow = studentGradeTable.getSelectedRow();
    if (selectRow < 0) {
        JOptionPane.showMessageDialog(this, "请选择需要修改的行!", "",
```





```
JOptionPane.WARNING_MESSAGE);
    return;
} else {
    final GradeBean grade = new GradeBean();
    grade.setId((Integer) studentGradeTable.getValueAt(selectRow, 0));
    grade.setStudentName((String) studentGradeTable.getValueAt(selectRow,
1));
    grade.setStudentClass((String) studentGradeTable.getValueAt(selectRow,
2));
    grade.setTestSubject((String) studentGradeTable.getValueAt(selectRow,
3));
    grade.setScore((String) studentGradeTable.getValueAt(selectRow, 4));
    grade.setTestTime((String) studentGradeTable.getValueAt(selectRow,
5));
    grade.setRemark((String) studentGradeTable.getValueAt(selectRow, 6));
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                GradeModificationFrame frame = new
GradeModificationFrame(studentGradeTable, grade);
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

在上面的代码中，首先获得用户选择的行，然后将该行中包含的数据保存到一个 GradeBean 对象中，接着将这个对象传递给 GradeModificationFrame 类。该类用于修改用户选择的记录，具体代码将在下节进行讲解。

## 2.2.4 “删除”按钮事件监听

“删除”按钮用于删除用户选择的成绩单，其关键代码如下：

```
protected void do_deleteButton_actionPerformed(ActionEvent e) {
    int selectRow = studentGradeTable.getSelectedRow();
    if (selectRow < 0) {
        JOptionPane.showMessageDialog(this, "请选择需要删除的行！", "",
JOptionPane.WARNING_MESSAGE);
        return;
    } else {
        final GradeBean grade = new GradeBean();
        grade.setId((Integer) studentGradeTable.getValueAt(selectRow, 0));
        JdbcHelper.delete(grade);
        DefaultTableModel model = (DefaultTableModel)
studentGradeTable.getModel(); // 获得表格模型
        model.setRowCount(0); // 清空表格中的数据
        model.setColumnIdentifiers(new Object[] { "编号", "学生姓名", "学生班级",
"考试科目", "考试成绩", "考试时间", "备注" });
        // 获得数据库中表格的全部数据
        List<GradeBean> results = JdbcHelper.queryAll();
        for (GradeBean tempGrade : results) { // 将数据加载到表格模型中
            model.addRow(new Object[] { tempGrade.getId(),
tempGrade.getStudentName(),
tempGrade.getStudentClass(),
tempGrade.getTestSubject(), tempGrade.getScore(),
tempGrade.getTestTime(),
tempGrade.getRemark() });
        }
    }
}
```







```

    }
    studentGradeTable.setModel(model);
}

```

在上面的代码中, 首先获得用户选择的行, 然后将该行的编号传递给 GradeBean 对象; 接着调用 JdbcHelper 类工具方法 delete() 来删除对应的记录; 最后重新加载表格中的数据, 完成刷新操作。

## 2.6 修改成绩单

### 2.6.1 功能概述

“修改成绩单”窗体用于成绩单修改。根据用户在图 2.14 中选择的行, 填充图 2.15 中的控件。单击“修改”按钮完成修改信息的保存。

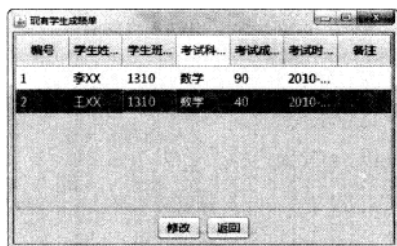


图 2.14 “现有学生成绩单”窗体

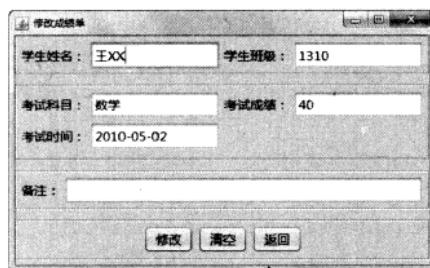


图 2.15 “修改成绩单”窗体

### 2.6.2 设置文本框控件

在图 2.15 中, 定义了很多文本框控件, 用于保存用户输入的信息。在打开该窗体时, 会根据数据库中保存的信息来填写这些控件, 定义控件的关键代码如下:

```

JPanel studentInfoPanel = new JPanel(); // 创建面板
studentInfoPanel.setBounds(0, 0, 484, 46); // 设置面板位置
studentInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
// 获得面板布局
FlowLayout flowLayout = (FlowLayout) studentInfoPanel.getLayout();
flowLayout.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(studentInfoPanel); // 应用面板

JLabel studentNameLabel = new JLabel("学生姓名: "); // 创建标签
studentNameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameLabel); // 应用标签

studentNameTextField = new JTextField(); // 创建文本框
studentNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameTextField); // 应用文本框
studentNameTextField.setColumns(10); // 设置文本框显示的列数

```







```
JLabel studentClassLabel = new JLabel("学生班级: "); // 创建标签
studentClassLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassLabel); // 应用标签

studentClassTextField = new JTextField(); // 创建文本框
studentClassTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassTextField); // 应用文本框
studentClassTextField.setColumns(10); // 设置文本框显示的列数

testInfoPanel = new JPanel(); // 创建面板
testInfoPanel.setBounds(0, 58, 484, 89); // 设置面板位置
testInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
// 获得面板布局
FlowLayout flowLayout_1 = (FlowLayout) testInfoPanel.getLayout();
flowLayout_1.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(testInfoPanel); // 应用面板

testSubjectLabel = new JLabel("考试科目: "); // 创建标签
testSubjectLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectLabel); // 应用标签

testSubjectTextField = new JTextField(); // 创建文本框
testSubjectTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectTextField); // 应用文本框
testSubjectTextField.setColumns(10); // 设置文本框显示的列数

scoreLabel = new JLabel("考试成绩: "); // 创建标签
scoreLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreLabel); // 应用标签

scoreTextField = new JTextField(); // 创建文本框
scoreTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreTextField); // 应用文本框
scoreTextField.setColumns(10); // 设置文本框显示的列数

testTimeLabel = new JLabel("考试时间: "); // 创建标签
testTimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeLabel); // 应用标签

testTimeTextField = new JTextField(); // 创建文本框
testTimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeTextField); // 应用文本框
testTimeTextField.setColumns(10); // 设置文本框显示的列数

remarkPanel = new JPanel(); // 创建面板
remarkPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
FlowLayout flowLayout_2 = (FlowLayout) remarkPanel.getLayout(); // 获得面板布局
flowLayout_2.setAlignment(FlowLayout.LEFT); // 设置面板中控件按左对齐排列
remarkPanel.setBounds(0, 159, 484, 46); // 设置面板位置
contentPane.add(remarkPanel); // 应用面板

JLabel remarkLabel = new JLabel("备注: "); // 创建标签
remarkLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkLabel); // 应用标签

remarkTextField = new JTextField(); // 创建文本框
remarkTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkTextField); // 应用文本框
remarkTextField.setColumns(29); // 设置文本框显示的列数
```







### 2.6.3 添加工具按钮

在窗体中使用了“修改”、“清空”和“返回”3个按钮，并将它们放置在一个面板中，其关键的代码如下：

```
JPanel buttonPanel = new JPanel();           // 创建面板
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
// 设置面板边框
buttonPanel.setBounds(0, 217, 484, 48);      // 设置面板位置
contentPane.add(buttonPanel);                // 应用面板

JButton modifyButton = new JButton("修改");   // 创建按钮
// 省略事件处理相关代码
modifyButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(modifyButton);                // 应用按钮

JButton clearButton = new JButton("清空");    // 创建按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton);                 // 应用按钮

JButton returnButton = new JButton("返回");   // 创建按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton);                // 应用按钮
```

### 2.6.4 填充成绩单信息

修改是在用户选择的行的基础上进行的，根据构造方法传递进来的参数，使用自定义的 updateContent() 方法添加成绩单信息到窗体，该方法的关键代码如下：

```
private void updateContent(GradeBean grade) {
    studentNameTextField.setText(grade.getStudentName());
    studentClassTextField.setText(grade.getStudentClass());
    testSubjectTextField.setText(grade.getTestSubject());
    scoreTextField.setText(grade.getScore());
    testTimeTextField.setText(grade.getTestTime());
    remarkTextField.setText(grade.getRemark());
}
```

### 2.6.5 修改成绩单信息

通过监听“修改”按钮单击事件，实现对用户修改的成绩单信息的保存功能。由于成绩单上的信息都非常重要，因此不能为空值（备注信息除外）。“修改”按钮事件监听器的关键代码如下：

```
modifyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_modifyButton_actionPerformed(e);
    }
});
```

do\_modifyButton\_actionPerformed() 方法是 IDE 工具生产的方法，它完成了对文本域的





非空校验及保存用户修改信息的功能，最后刷新表格，其关键代码如下：

```
protected void do_modifyButton_actionPerformed(ActionEvent e) {
    String studentName = studentNameTextField.getText().trim();
    if (studentName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "学生姓名不能为空！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String studentClass = studentClassTextField.getText().trim();
    if (studentClass.isEmpty()) {
        JOptionPane.showMessageDialog(this, "学生班级不能为空！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (!ValidationUtil.validateStudentClass(studentClass)) {
        JOptionPane.showMessageDialog(this, "学生班级为 0102 样式！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String testSubject = testSubjectTextField.getText().trim();
    if (testSubject.isEmpty()) {
        JOptionPane.showMessageDialog(this, "考试科目不能为空！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String score = scoreTextField.getText().trim();
    if (score.isEmpty()) {
        JOptionPane.showMessageDialog(this, "考试成绩不能为空！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (!ValidationUtil.validateScore(score)) {
        JOptionPane.showMessageDialog(this, "考试成绩为 88 样式！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String testTime = testTimeTextField.getText().trim();
    if (testTime.isEmpty()) {
        JOptionPane.showMessageDialog(this, "考试时间不能为空！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (!ValidationUtil.validateTestTimeFormat(testTime)) {
        JOptionPane.showMessageDialog(this, "考试时间为 2011-02-23 样式！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String remark = remarkTextField.getText().trim();
    grade.setStudentName(studentName);
    grade.setStudentClass(studentClass);
    grade.setTestSubject(testSubject);
    grade.setScore(score);
    grade.setTestTime(testTime);
    grade.setRemark(remark);
    int result = JdbcHelper.update(grade);
    if (result >= 0) {
        JOptionPane.showMessageDialog(this, "成绩单修改成功！");
        DefaultTableModel model = (DefaultTableModel)
studentGradeTable.getModel(); // 获得表格模型
        model.setRowCount(0); // 清空表格中的数据
        model.setColumnIdentifiers(new Object[] { "编号", "学生姓名", "学生班级",
"考试科目",
"考试成绩", "考试时间", "备注" });
        // 获得数据库中表格的全部数据
        List<GradeBean> results = JdbcHelper.queryAll();
    }
}
```







```

        for (GradeBean grade : results) { // 将数据加载到表格模型中
            model.addRow(new Object[] { grade.getId(), grade.getStudentName(),
                grade.getStudentClass(),
                grade.getTestSubject(),
                grade.getScore(), grade.getTestTime(), grade.getRemark() });
        }
        studentGradeTable.setModel(model);
        return;
    } else {
        JOptionPane.showMessageDialog(this, "成绩单修改失败!");
        return;
    }
}

```

## 2.7 查询成绩单

### 2.7.1 功能概述

为了方便用户查询以前保存的成绩单, 本模块提供了查询功能。用户可以使用学生姓名、学生班级、考试科目、考试成绩、考试时间和备注中任意一项或几项进行模糊查询。单击主窗体“成绩单管理”/“查询成绩单”菜单项, 就可以打开“查询成绩单”窗体, 如图 2.16 所示。

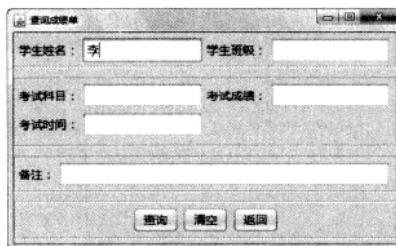


图 2.16 “查询成绩单”窗体

### 2.7.2 设置文本框控件

在图 2.16 中, 定义了很多文本框控件用于接收用户输入的查询信息, 用户可以使用各种不同的条件实现模糊查询。为了方便用户, 使用边框将信息进行分类, 其关键代码如下:

```

JPanel studentInfoPanel = new JPanel(); // 创建面板
studentInfoPanel.setBounds(0, 0, 484, 46); // 设置面板位置
studentInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null,
    null)); // 设置面板边框
FlowLayout flowLayout = (FlowLayout) studentInfoPanel.getLayout();
// 获得面板布局
flowLayout.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(studentInfoPanel); // 应用面板

JLabel studentNameLabel = new JLabel("学生姓名:"); // 创建标签
studentNameLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameLabel); // 应用标签

studentNameTextField = new JTextField(); // 创建文本框
studentNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentNameTextField); // 应用文本框
studentNameTextField.setColumns(10); // 设置文本框显示的列数

JLabel studentClassLabel = new JLabel("学生班级:"); // 创建标签

```







```
studentClassLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassLabel); // 应用标签

studentClassTextField = new JTextField(); // 创建文本框
studentClassTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
studentInfoPanel.add(studentClassTextField); // 应用文本框
studentClassTextField.setColumns(10); // 设置文本框显示的列数

testInfoPanel = new JPanel(); // 创建面板
testInfoPanel.setBounds(0, 58, 484, 89); // 设置面板位置
testInfoPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
FlowLayout flowLayout_1 = (FlowLayout) testInfoPanel.getLayout(); // 获得面板布局
flowLayout_1.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
contentPane.add(testInfoPanel); // 应用面板

testSubjectLabel = new JLabel("考试科目:"); // 创建标签
testSubjectLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectLabel); // 应用标签

testSubjectTextField = new JTextField(); // 创建文本框
testSubjectTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testSubjectTextField); // 应用文本框
testSubjectTextField.setColumns(10); // 设置文本框显示的列数

scoreLabel = new JLabel("考试成绩:"); // 创建标签
scoreLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreLabel); // 应用标签

scoreTextField = new JTextField(); // 创建文本框
scoreTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(scoreTextField); // 应用文本框
scoreTextField.setColumns(10); // 设置文本框显示的列数

testTimeLabel = new JLabel("考试时间:"); // 创建标签
testTimeLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeLabel); // 应用标签

testTimeTextField = new JTextField(); // 创建文本框
testTimeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
testInfoPanel.add(testTimeTextField); // 应用文本框
testTimeTextField.setColumns(10); // 设置文本框显示的列数

remarkPanel = new JPanel(); // 创建面板
remarkPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null)); // 设置面板边框
FlowLayout flowLayout_2 = (FlowLayout) remarkPanel.getLayout(); // 获得面板布局
flowLayout_2.setAlignment(FlowLayout.LEFT); // 设置面板中的控件按左对齐排列
remarkPanel.setBounds(0, 159, 484, 46); // 设置面板位置
contentPane.add(remarkPanel); // 应用面板

JLabel remarkLabel = new JLabel("备注:"); // 创建标签
remarkLabel.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkLabel); // 应用标签
remarkTextField = new JTextField(); // 创建文本框
remarkTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
remarkPanel.add(remarkTextField); // 应用文本框
remarkTextField.setColumns(29); // 设置文本框显示的列数
```



### 2.7.3 添加工具按钮

在窗体中定义了“查询”、“清空”和“返回”3个按钮，其关键的代码如下：





```

JPanel buttonPanel = new JPanel();           // 创建面板
// 设置面板的边框
buttonPanel.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用面板

JButton queryButton = new JButton("查询");    // 创建“查询”按钮
// 省略事件处理相关代码
queryButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(queryButton);                // 应用按钮

JButton clearButton = new JButton("清空");    // 创建“清空”按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(clearButton);                // 应用按钮

JButton returnButton = new JButton("返回");   // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置字体
buttonPanel.add(returnButton);               // 应用按钮

```

## 2.7.4 查询成绩单信息

通过监听“查询”按钮单击事件，完成对用户输入条件的查询功能，这里要求用户至少输入一项查询条件。如果查询到结果，将在一个新窗体中进行显示，如图 2.17 所示；否则提示用户结果并不存在，如图 2.18 所示。

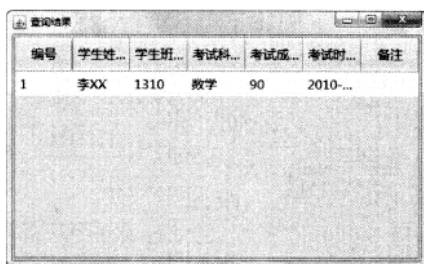


图 2.17 查询成功时的窗体



图 2.18 查询失败时的窗体

“查询”按钮事件监听器的关键代码如下：

```

queryButton.addActionListener(new ActionListener() { // 监听按钮事件
    @Override
    public void actionPerformed(ActionEvent e) {
        do_queryButton_actionPerformed(e);
    }
});

```

do\_queryButton\_actionPerformed()方法是 IDE 工具生产的方法，其关键代码如下：

```

protected void do_queryButton_actionPerformed(ActionEvent e) {
    String studentName = studentNameTextField.getText().trim();
    String studentClass = studentClassTextField.getText().trim();
    String testSubject = testSubjectTextField.getText().trim();
    String score = scoreTextField.getText().trim();
    String testTime = testTimeTextField.getText().trim();
    String remark = remarkTextField.getText().trim();
    if (studentName.isEmpty() && studentClass.isEmpty() &&
        testSubject.isEmpty()
        && score.isEmpty() && testTime.isEmpty() && remark.isEmpty()) {

```







```
JOptionPane.showMessageDialog(this, "查询条件不能为空!", "",
JOptionPane.WARNING_MESSAGE);
return;
}

if ((!studentClass.isEmpty()) &&
(!ValidationUtil.validateStudentClass(studentClass))) {
JOptionPane.showMessageDialog(this, "学生班级为 0102 样式!", "",
JOptionPane.WARNING_MESSAGE);
return;
}

if ((!score.isEmpty()) && (!ValidationUtil.validateScore(score))) {
JOptionPane.showMessageDialog(this, "考试成绩为 88 样式!", "",
JOptionPane.WARNING_MESSAGE);
return;
}

if ((!testTime.isEmpty()) &&
(!ValidationUtil.validateTestTimeFormat(testTime))) {
JOptionPane.showMessageDialog(this, "考试时间为 2011-02-23 样式!", "",
JOptionPane.WARNING_MESSAGE);
return;
}

GradeBean grade = new GradeBean();
grade.setStudentName(studentName);
grade.setStudentClass(studentClass);
grade.setTestSubject(testSubject);
grade.setScore(score);
grade.setTestTime(testTime);
grade.setRemark(remark);
final List<GradeBean> results = JdbcHelper.query(grade);
if (results.size() > 0) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                GradeQueryResultFrame frame = new
GradeQueryResultFrame(results);
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
    return;
} else {
    JOptionPane.showMessageDialog(this, "无符合条件的结果!");
    return;
}
}
```

### 2.7.5 显示查询结果



62



在完成查询后，将结果保存在一个 List 控件中，然后将其作为参数传递到 GradeQueryResultFrame 类的构造方法中，该类使用一个表格控件来显示查询结果，其关键代码如下：

```
public GradeQueryResultFrame(List<GradeBean> results) {
    setTitle("查询结果");
    contentPane = new JPanel();
}
```





```

contentPane.setLayout(new BorderLayout(0, 0));
setContentPane(contentPane);

JScrollPane scrollPane = new JScrollPane();
contentPane.add(scrollPane, BorderLayout.CENTER);

studentGradeTable = new JTable(); // 创建表格
// 设置表体字体
studentGradeTable.setFont(new Font("微软雅黑", Font.PLAIN, 15));
studentGradeTable.setRowHeight(30); // 设置表体高度
JTableHeader header = studentGradeTable.getTableHeader(); // 获得表头对象
header.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置表头字体
// 设置表头高度
header.setPreferredSize(new Dimension(header.getWidth(), 40));

DefaultTableCellRenderer renderer = (DefaultTableCellRenderer)
header.getDefaultRenderer(); // 获得渲染器
// 设置表头内容居中显示
renderer.setHorizontalAlignment(SwingConstants.CENTER);

DefaultTableModel model = (DefaultTableModel)
studentGradeTable.getModel(); // 获得表格模型
model.setRowCount(0); // 清空表格中的数据
model.setColumnIdentifiers(new Object[] { "编号", "学生姓名", "学生班级", "
考试科目",
        "考试成绩", "考试时间", "备注" });
for (GradeBean grade : results) { // 将数据加载到表格模型中
    model.addRow(new Object[] { grade.getId(), grade.getStudentName(),
grade.getStudentClass(),
        grade.getTestSubject(), grade.getScore(),
grade.getTestTime(), grade.getRemark() });
}
studentGradeTable.setModel(model); // 应用表格模型
scrollPane.setViewportView(studentGradeTable);
setLocation(WindowUtil.getLocation()); // 设置窗体显示位置
setSize(WindowUtil.getSize()); // 设置窗体大小
}

```



# 第 3 章

---

## 常用照片管理模块

(Swing+图片处理技术实现)

---

由于数码相机比传统相机具有难以比拟的优势，例如便于照片的传播、价格相对低廉等，加速了数码相机的普及。但是，如何管理大量的照片又成了一个难题。目前很多软件中都提供了对数码照片的管理功能，基于这一点，本章将实现一个可重用的常用照片管理模块，用于人们管理照片、图片等，方便程序人员开发类似程序。通过本章的学习，读者能够学到：

- » 缩略图浏览方式的实现方法
- » 幻灯片浏览方式的实现方法
- » 播放器浏览方式的实现方法
- » 实用的搜索照片的方法
- » 树控件常用方法





## 3.1 常用照片管理模块概述

### 3.1.1 设计思路

常用照片管理模块将设计成一个可重用的模块,软件开发人员只要将该模块添加到自己的系统中,就实现了数码照片的管理功能。在该模块中,主要提供相册管理、照片及其信息管理、浏览照片和查询照片4大功能,下面将逐一介绍它们的设计思路。

相册展开后将呈现为树状结构,即在相册中还可以包含子相册,这样可以对照片进行详细分类,实现对照片的梯级管理。选中相册后,可以对其进行修改和删除,也可以向其中添加子相册,如果在添加时未选中任何相册,则表示添加的是最顶层相册。

选中要查看的相册后,在照片显示区将显示出该相册中的所有照片,浏览照片的方式共有3种:缩略图方式、幻灯片方式和播放器方式,其中缩略图方式为默认的浏览方式。在缩略图和幻灯片方式下,双击照片可以使照片全屏显示,双击鼠标或按【Esc】键可以退出全屏显示;在播放器方式中将提供两种播放模式,分别为手动和自动模式,在自动模式下可以修改每张照片的展示时间,也可以通过将光标移动到照片上使播放暂停,离开后将继续播放。

在缩略图和幻灯片方式下,可以对照片进行管理,包括添加、修改和删除照片。支持批量添加和删除照片,即如果在添加过程中选择的是文件路径(文件夹),将添加该路径下的所有照片到当前相册中;在删除照片之前需要先选中要删除的照片,通过配合使用鼠标和键盘按键(【Ctrl】键和【Shift】键),可以同时选中多个照片。

在查找照片时,可以根据照片的标题、拍摄日期和描述进行查找,其中拍摄日期可以是以一个时间点为界限进行查找,也可以指定具体的时间段进行查找,符合查询条件的照片将显示在照片显示区,单击照片可以查看其所属的相册。

### 3.1.2 功能结构

根据常用照片管理模块的设计思路,可以将该模块分为相册管理和照片管理两部分来实现,具体的模块架构如图3.1所示。



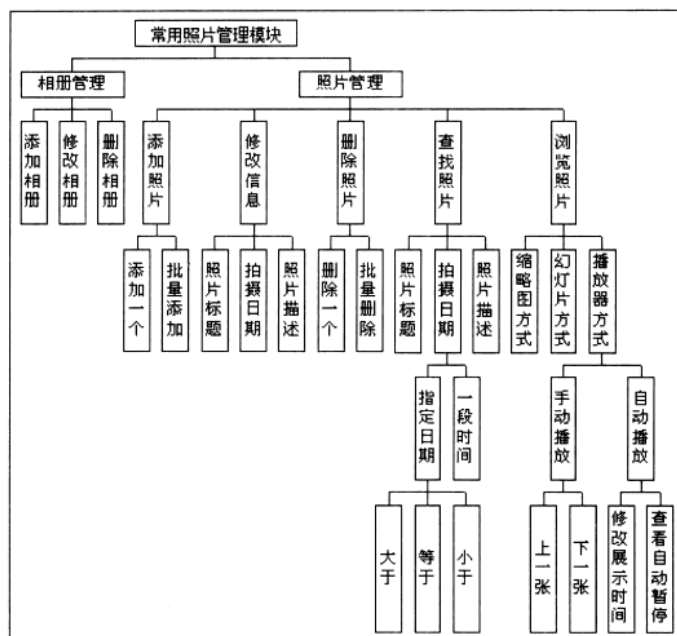


图 3.1 常用照片管理模块功能结构

### 3.1.3 效果预览

图 3.2 是以缩略图方式浏览指定相册中的照片,并且演示了同时选中多个照片的功能。



图 3.2 缩略图浏览方式及选中照片







图 3.3 所示为以幻灯片方式浏览指定相册中的照片，并且展示了快速查看照片信息的功能。



图 3.3 幻灯片浏览方式及查看照片信息

图 3.4 所示为以播放器方式浏览指定相册中的照片，该方式将在一个无标题栏的对话框中实现。

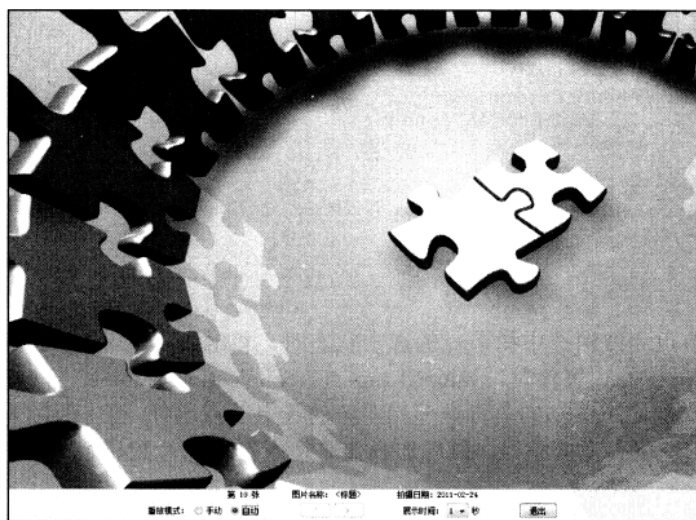


图 3.4 播放器浏览方式

图 3.5 所示为修改照片信息的对话框，可以为照片添加标题、拍摄日期和描述。

图 3.6 所示为填写查找条件的对话框，照片的标题、拍摄日期和描述均可以作为查找条件进行查找。



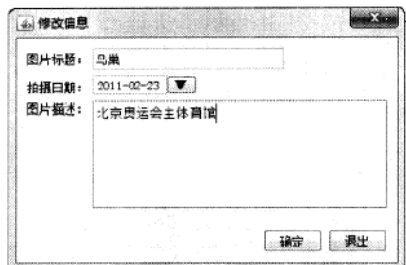


图 3.5 修改信息



图 3.6 查找照片

## 3.2 关键技术

在开发常用照片管理模块的过程中，将用到一些关键技术，它们对本模块的开发至关重要。为了帮助读者在正式开发前扫除障碍，下面将对这些技术进行详细讲解。

### 3.2.1 捕获树的选中节点事件

如果要查看某一相册中的照片，就要选中该相册，即选中相应的树节点。当节点被选中后，将加载其直接包含的照片到显示区域，为相册树添加捕获选中节点事件监听器的关键代码如下：

```
albumTree.addTreeSelectionListener(new
javax.swing.event.TreeSelectionListener() {
    public void valueChanged(javax.swing.event.TreeSelectionEvent evt) {
        try {
            albumTreeViewValueChanged(evt);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
});
```

当选中树节点和取消选中树节点的选中状态时，将触发 `TreeSelectionEvent` 事件。当捕获 `TreeSelectionEvent` 事件时，`valueChanged(TreeSelectionEvent e)` 方法将被触发执行。方法 `albumTreeViewValueChanged()` 负责处理相册树的选中节点事件，首先清空幻灯片标签和图片箱，然后获得选中节点的路径，最后加载该路径下所有照片到显示区域，该方法的关键代码如下：

```
private void albumTreeViewValueChanged(TreeSelectionEvent evt) throws
UnsupportedEncodingException {
    // 存在加载图片的线程
    if (loadPhotoThread != null && loadPhotoThread.isAlive()) {
        synchronized (loadPhotoThread) { // 将其加入到同步块中
            loadPhotoThread.interrupt(); // 中断线程
        }
    }
    // 获得浏览方式面板
    final MPanel showPanel = (MPanel) photoPanel.getComponent(0);
    final MLabel photoLabel = showPanel.getShowPhotoLabel(); // 获得幻灯片标签
    if (photoLabel.getIcon() != null) { // 幻灯片方式
```





```

        photoLabel.setIcon(null); // 清空幻灯片
    }
    final JPanel photoBoxPanel = showPanel.getPhotoBoxPanel(); // 获得图片箱面板
    if (photoBoxPanel.getComponentCount() > 0) { // 如果图片箱不为空
        photoBoxPanel.removeAll(); // 清空图片箱
    }
    String selectedPath = getSelectedPath(); // 获得选中相册的路径
    if (selectedPath == null) { // 如果路径为空
        showPanel.validate(); // 刷新面板
    } else { // 如果路径不为空
        // 获得所有照片对象
        final File[] photos = new File(selectedPath).listFiles(ioFileFilter);
        if (photos != null && photos.length > 0) { // 如果有照片
            // 添加第一个照片到图片箱
            photoBoxPanel.add(new PhotoPreviewButton(photos[0]));
            if (photoLabel != null) { // 幻灯片方式
                // 为幻灯片设置照片
                photoLabel.setIcon(new
                    ImageIcon(photos[0].getPath())); // 刷新面板
                showPanel.validate(); // 刷新面板
                loadPhotoThread = new Thread() { // 创建一个用来加载照片到图片箱的线程
                    @Override
                    public void run() { // 重载该方法
                        for (int i = 1; i < photos.length; i++) { // 遍历照片数组
                            try {
                                Thread.sleep(600); // 休眠6秒
                            } catch (InterruptedException e) {
                                break;
                            }
                            // 添加指定照片到图片箱
                            photoBoxPanel.add(new PhotoPreviewButton(photos[i]));
                            showPanel.validate(); // 刷新面板
                        }
                    }
                };
                loadPhotoThread.start(); // 开启线程
            } else { // 如果没有照片
                showPanel.validate(); // 刷新面板
            }
        }
    }
}

```



注意 在加载照片的线程中刷新面板时，不要直接刷新包含照片的面板，那样将无

法刷新滚动条。

### 3.2.2 捕获树的展开节点事件

在系统刚刚启动时，并没有加载整个相册树到系统中，而是只加载了顶级相册。每当用户初次展开相册节点时，才加载该节点包含的子节点，这样就需要捕获相册节点展开的事件，为相册树添加捕获展开节点事件监听器的关键代码如下：

```

albumTree.addTreeExpansionListener(new
    javax.swing.event.TreeExpansionListener() {
        @Override
        public void treeCollapsed(javax.swing.event.TreeExpansionEvent evt) {
        }
    }
);

```





```
@Override
public void treeExpanded(javax.swing.event.TreeExpansionEvent evt) {
    albumTreeTreeExpanded(evt);
}
});
```

如果此事件是由节点展开发出的, 方法 `treeExpanded()` 将被触发; 如果此事件是由节点折叠发出的, 方法 `treeCollapsed()` 将被触发。这里只关注节点展开的事件, 方法 `albumTreeTreeExpanded()` 负责处理相册树的展开节点事件, 首先获得展开节点的对象, 然后查看该节点的子节点是否已经被加载, 如果尚未加载则加载它的子节点, 并刷新树模型, 该方法的关键代码如下:

```
private void albumTreeTreeExpanded(javax.swing.event.TreeExpansionEvent evt) {
    TreePath selectedPath = evt.getPath(); // 获得选中节点的路径对象
    // 获得选中节点对象
    MTreeNode lastNode = (MTreeNode) selectedPath.getLastPathComponent();
    if (!lastNode.isLoad()) { // 如果该节点尚未加载
        loadChildNode(lastNode); // 加载该节点
        treeModel.reload(lastNode); // 刷新树模型
    }
}
```

### 3.2.3 浏览方式切换技术

在实现照片浏览功能时, 设计了 3 种方式, 分别为缩略图、幻灯片和播放器方式。其中缩略图和幻灯片方式将在照片显示区实现, 具体的效果如图 3.7 和图 3.8 所示。这部分的难点在于如何快速在几种浏览方式间切换。

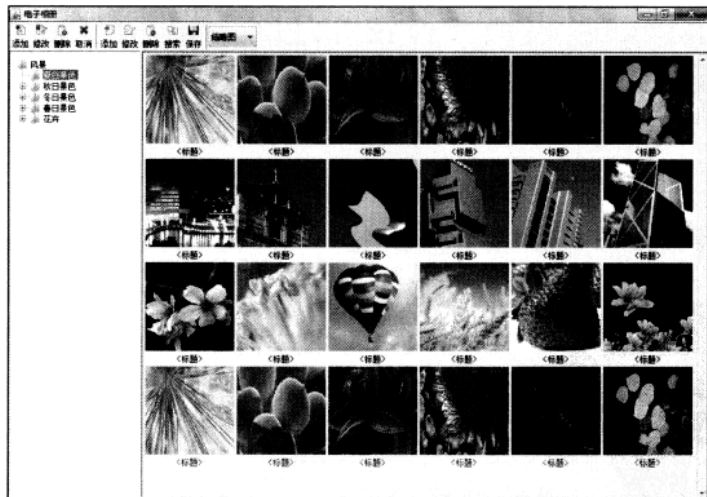


图 3.7 缩略图浏览方式

首先讲解一下这两种浏览方式的实现过程。对于缩略图浏览方式, 可以将照片显示区采用 6 列的网格布局; 对于幻灯片浏览方式, 可以将照片显示区采用边框布局。在幻灯片浏览方式中, 照片显示区下面有一个用于选择显示图片的面板, 其代码如下:





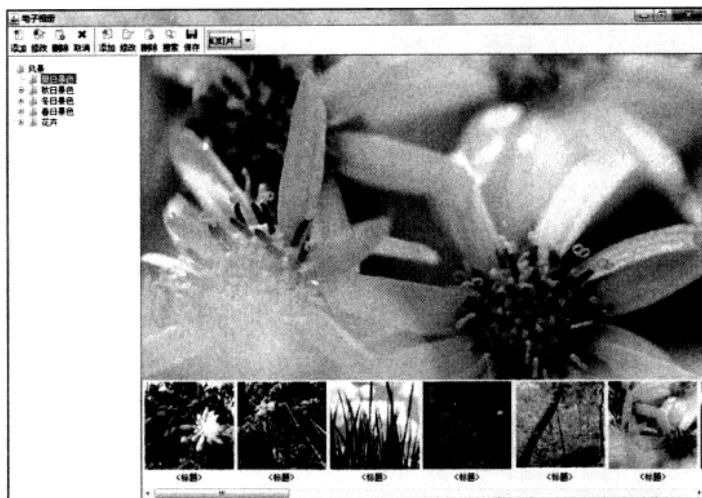


图 3.8 幻灯片浏览方式

```
public class MPanel extends JPanel {
    private static final long serialVersionUID = 491131278474248447L;
    protected static final JPanel photoBoxPanel = new JPanel(); // 图片箱面板
    protected static final JLabel showPhotoLabel = new JLabel(); // 幻灯片标签
    public MPanel() {
        setLayout(new BorderLayout()); // 采用边框式布局
    }
    public JPanel getPhotoBoxPanel() {
        return photoBoxPanel;
    }
    public JLabel getShowPhotoLabel() {
        return showPhotoLabel;
    }
}
```

在切换浏览方式时,使用的是 JComboBox 控件。为其增加的监听器调用了自定义的方法 seeModeComboBoxItemStateChanged(),它利用 switch 语句来更改浏览方式。在更换浏览方式时,首先要去掉当前使用的面板,最后还要刷新面板,该方法关键代码如下:

```
private void seeModeComboBoxItemStateChanged(java.awt.event.ItemEvent evt) {
    if (evt.getStateChange() == ItemEvent.SELECTED) { // 由选中新的项目触发
        // 获得选中项的索引值
        int selectedIndex = seeModeComboBox.getSelectedIndex();
        switch (selectedIndex) {
            case 0: // 缩略图方式
                photoPanel.remove(0); // 移除幻灯片方式面板
                photoPanel.add(new BreviaryPhotoPanel()); // 添加缩略图方式面板
                photoPanel.validate(); // 刷新面板
                break;
            case 1: // 幻灯片方式
                photoPanel.remove(0); // 移除缩略图方式面板
                // 创建幻灯片方式面板
                LanternSlidePanel panel = new LanternSlidePanel();
                // 如果存在照片
                if (panel.getPhotoBoxPanel().getComponentCount() > 0) {
                    PhotoPreviewButton lanternSlide = null; // 幻灯片
                    // 如果存在被选中的照片
                    if (PhotoPreviewButton.getSelectedPhoto().size() > 0) {
                        // 获得最后一次单击的照片
                        lanternSlide = PhotoPreviewButton.getSelectedPhoto().
                            lastElement();
                    }
                }
            }
        }
    }
}
```





```
    } else { // 不存在被选中的照片
        // 获得第一个照片
        lanternSlide = (PhotoPreviewButton)
            panel.getPhotoBoxPanel().getComponent(0);
    }
    panel.getShowPhotoLabel().setIcon(new
        ImageIcon(lanternSlide.getPath())); // 设置幻灯片
    }
    photoPanel.add(panel); // 添加幻灯片方式面板
    photoPanel.validate(); // 刷新面板
    break;
default: // 播放器方式
    // 获得浏览方式面板
    MPanel showPanel = (MPanel) photoPanel.getComponent(0);
    Component[] photos =
        showPanel.getPhotoBoxPanel().getComponents(); // 获得照片数组
    if (photos.length > 0) { // 如果存在照片
        // 显示播放器对话框
        new PlayDialog(null, true,
            photos).setVisible(true);
        // 移除选项事件监听器
        seeModeComboBox.removeItemListener(showModeComboBoxIL);
        // 设置之前的选中项仍被选中
        seeModeComboBox.setSelectedItem(primaryItem);
        // 添加选项事件监听器
        seeModeComboBox.addItemListener(showModeComboBoxIL);
    }
} else { // 由取消原选中项触发
    primaryItem = evt.getItem(); // 获得原选中项
}
}
```

### 3.2.4 随意选取照片技术

在实现批量删除照片时，关键是如何解决随意选取照片的问题，设计的选取方式共有 3 种，具体的选取方式如下：

通过单击鼠标按钮选取，通过这种方式每次只能选取一个，并且在同一时间只能有一个处于被选取状态，效果如图 3.9 所示。

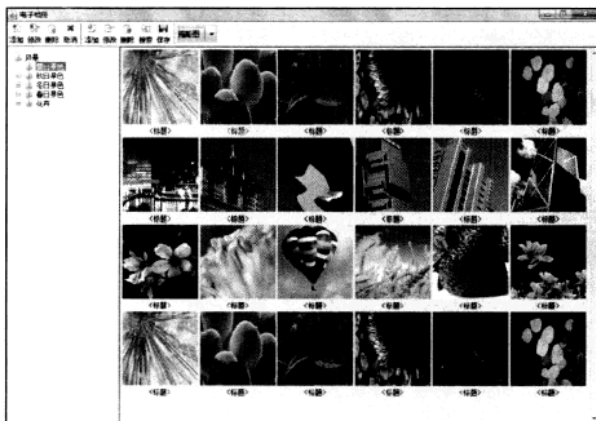


图 3.9 单击鼠标按钮选取







在按下【Ctrl】键的情况下通过单击鼠标按钮选取,通过这种方式每次只能选取一个,但是之前被选取的将依然处于被选取状态,即在同一时间可以有多个处于被选取状态,效果如图 3.10 所示。

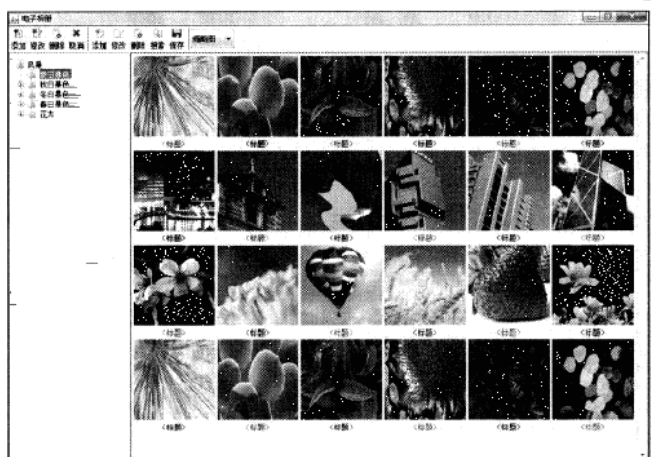


图 3.10 按下【Ctrl】键选取

在按下【Shift】键的情况下通过单击鼠标按钮选取,通过这种方式每次可以选取多个,被选取的对象为上次和本次用鼠标单击的对象,以及所有位于它们之间的对象,效果如图 3.11 所示。

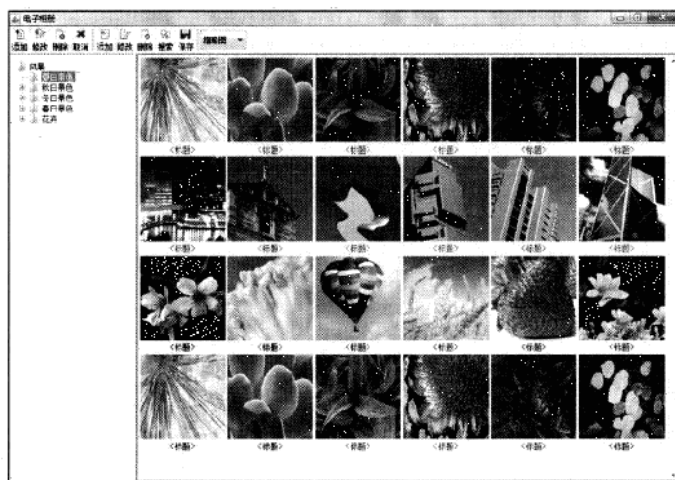


图 3.11 按下【Shift】键选取

通过对上面 3 种选取方式的配合使用,就可以随意选取照片了。在缩略图和幻灯片浏览方式中的缩略图,是显示在按钮控件上的,所以可以通过捕获按钮被按下事件,处理照片的选取动作。不过在处理选取动作之前,还需要判断是否有【Ctrl】键或【Shift】键被按下,可以通过捕获按钮的键盘事件,获得最后被按下的按键的键值。为按钮添加键盘





事件监听器的关键代码如下：

```
addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {           // 当键盘按键被按下时触发
        keyCode = e.getKeyCode();                 // 获得被按下的按键的键值
    }
    @Override
    public void keyReleased(KeyEvent e) {          // 当键盘按键被释放时触发
        keyCode = 0;                             // 恢复为默认键值
    }
});
```

当有按键被按下时方法 `keyPressed()` 将被触发，此时获得被按下的按键的键值；当有按键被释放时方法 `keyReleased()` 将被触发，此时键值将恢复为默认值。为按钮添加动作事件监听器的关键代码如下：

```
addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获得当前按钮对象
        PhotoPreviewButton noncePhoto = PhotoPreviewButton.this;
        switch (keyCode) {
            // 判断当前按下的键
            case KeyEvent.VK_CONTROL:              // 【Ctrl】键被按下，追加选中当前按钮
                noncePhoto.setForeground(Color.RED); // 设置前景色
                selectedPhoto.add(noncePhoto);      // 添加到向量中
                break;
            // 【Shift】键被按下，追加选中上次按下的按钮与当前按钮之间的所有按钮
            case KeyEvent.VK_SHIFT:
                if (selectedPhoto.isEmpty()) { // 尚未选中任何照片
                    noncePhoto.setForeground(Color.RED); // 设置前景色
                    selectedPhoto.add(noncePhoto);      // 添加到向量中
                } else {
                    // 获得上次按下的按钮对象
                    JButton lastPhoto = selectedPhoto.lastElement();
                    cancelSelected(); // 清空被选中的图片按钮
                    JButton startPhoto = lastPhoto; // 开始的图片按钮对象
                    JButton endPhoto = noncePhoto; // 结束的图片按钮对象
                    // 获得上次按下按钮的起始绘制点坐标
                    Point lastLocation = lastPhoto.getLocation();
                    // 获得当前按钮的起始绘制点坐标
                    Point nonceLocation = noncePhoto.getLocation();
                    // 确定上次按下的按钮和本次按下的按钮的具体顺序
                    if (nonceLocation.getY() == lastLocation.getY()) { // 在同一行
                        // 本次按下的按钮在前
                        if (nonceLocation.getX() < lastLocation.getX()) {
                            startPhoto = noncePhoto;
                            endPhoto = lastPhoto;
                        }
                    } else { // 不在同一行
                        // 本次按下的按钮在前
                        if (nonceLocation.getY() < lastLocation.getY()) {
                            startPhoto = noncePhoto;
                            endPhoto = lastPhoto;
                        }
                    }
                }
                // 获得所有图片按钮对象
                Component[] components = PhotoPreviewButton.this.getParent().
getComponents();
                boolean isSelected = false; // 默认为未遍历到第一个被选中的图片按钮
                for (int i = 0; i < components.length; i++) { // 遍历图片按钮数组
                    PhotoPreviewButton button = (PhotoPreviewButton)
components[i]; // 获得图片按钮对象
                    if (isSelected) { // 已经遍历到了第一个被选中的图片按钮
                        button.setForeground(Color.RED); // 设置前景色
                        selectedPhoto.add(button); // 添加到向量中
                    }
                }
            }
        }
    }
});
```







```

// 已经遍历到了最后一个被选中的图片按钮
if (button.equals(endPhoto)) {
    break; // 停止遍历, 跳出循环
}
} else { // 尚未遍历到了第一个被选中的图片按钮
// 已经遍历到了第一个被选中的图片按钮
if (button.equals(startPhoto)) {
    button.setForeground(Color.RED); // 设置前景色
    selectedPhoto.add(button); // 添加到向量中
    isSelected = true; // 已经遍历到了第一个被选中的图片按钮
}
}
// 将本次按下的图片按钮对象再次添加到向量的最后
selectedPhoto.add(noncePhoto);
}
break;
default: // 未按下任何键
cancelSelected(); // 清空被选中的图片按钮
noncePhoto.setForeground(Color.RED); // 设置前景色
selectedPhoto.add(noncePhoto); // 添加到向量中
}
}

```



**说明** 方法 getLocation() 返回的是一个 Point 类型的对象, 代表一个点的坐标。如

果控件是从左到右绘制的, 则为控件左上角的点的坐标, 如果控件是从右到左绘制的, 则为控件右上角的点的坐标。

### 3.2.5 照片缩放与内存溢出

在查看相册中的照片时, 将在短时间内生成大量按钮控件对象及照片缩略图, 这是一项极其耗费内存的操作。如果不能很好地解决该问题, 将是后续开发的一大障碍。

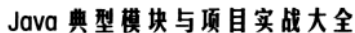
可以在按钮大小被调整时生成照片的缩略图, 并将其设置为按钮背景。通过为按钮控件添加 ComponentListener 监听器, 可以捕获按钮大小被调整事件, 这样就可以解决内存溢出问题。最后将该监听器移除, 为按钮添加 ComponentListener 监听器的关键代码如下:

```

addComponentListener(new ComponentAdapter() {
    @Override
    public void componentResized(ComponentEvent e) {
        // 创建 ImageIcon 类型的图片对象
        ImageIcon imageIcon = new ImageIcon(photoFile.getPath());
        Image image = createImage(130, 130); // 创建指定大小的 Image 类型的对象
        Graphics g = image.getGraphics(); // 获得 image 的绘图对象
        g.drawImage(imageIcon.getImage(), 0, 0, 130, 130,
            PhotoPreviewButton.this); // 指定绘图图片到 image
        image.flush(); // 刷新 image
        setIcon(new ImageIcon(image)); // 设置图片
        Vector photoV = dao.selectPhoto(photoFile.getName()); // 获得图片信息
        setText(photoV.get(3).toString()); // 设置文本
        ToolTip.set(PhotoPreviewButton.this, photoV); // 设置工具提示
        setName(photoFile.getName()); // 设置图片名称
        setPath(photoFile.getPath()); // 设置图片路径
        removeComponentListener(this); // 移除该监听器
    }
});

```





如果将光标移动到照片的上方，会弹出该照片的相关信息，包括照片的主题、拍摄日期和描述等，当信息过长时将换行显示。

```
public class ToolTip {  
private static final String TITLE = "图片标题: ";  
private static final String DATE = "<br>拍摄日期: ";  
private static final String NOTE = "<br>图片描述: ";  
private static final String SPACE = "  
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&";  
  
    public static void set(JComponent comp, Vector photoV) {  
        StringBuffer toolTip = new StringBuffer();           // 创建工具提示对象  
        toolTip.append("<html>");                             // 添加 HTML 标签头  
        toolTip.append(TITLE);                                // 添加标题标签  
        toolTip.append(photoV.get(3));                        // 添加标题内容  
        toolTip.append(DATE);                                 // 添加日期标签  
        toolTip.append(photoV.get(2));                         // 添加日期内容  
        toolTip.append(NOTE);                                  // 添加描述标签  
        String note = photoV.get(4).toString();               // 获得描述内容  
        StringBuffer rowBuf = new StringBuffer();             // 创建工具提示行对象  
        int rowBytes = 0;                                       // 行字节数  
        String c = "";                                           // 字符  
        for (int i = 0; i < note.length(); i++) {  
            c = note.substring(i, i + 1);                     // 获得字符  
            rowBuf.append(c);                                   // 添加到工具提示行  
            rowBytes += (c.getBytes()[0] > 0 ? 1 : 2);         // 计算行字节数  
            if (rowBytes > 18) {                               // 如果行字节数大于 18  
                toolTip.append(rowBuf.toString());              // 添加工具提示行到工具提示  
                toolTip.append(SPACE);                          // 添加回行空格  
                rowBuf = new StringBuffer();                   // 重建工具提示行对象  
                rowBytes = 0;                                    // 恢复行字节数  
            }  
        }  
        if (rowBytes == 0) {  
            if (note.length() != 0) {  
                int length = toolTip.length();  
                // 移除最后添加的回行空格  
                toolTip.delete(length - SPACE.length(), length);  
            }  
        } else {                                              // 最后一行工具提示尚未添加  
            toolTip.append(rowBuf.toString());                 // 添加工具提示行到工具提示  
        }  
        toolTip.append("</html>");                              // 添加 HTML 标签尾  
        comp.setToolTipText(toolTip.toString());              // 设置工具提示  
    }  
}
```

在程序运行后，左侧显示的是相册树。相册树可以进行添加新相册、修改相册名称、





删除相册等操作。在添加新相册时, 如果存在被选中的相册, 则添加新相册到该相册中, 否则表示添加一个顶级相册。其运行效果如图 3.12 所示。

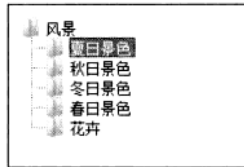


图 3.12 相册树显示效果

### 3.3.2 添加相册

单击工具栏菜单的第一个“添加”按钮可以完成添加相册的功能。该功能是通过 addAlbumButtonActionPerformed() 方法完成的, 该方法的关键代码如下:

```
private void addAlbumButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 获得被选中的相册节点
    MTreeNode lastSelectedNode = (MTreeNode)
albumTree.getLastSelectedPathComponent();
    int fatherId = (lastSelectedNode == null ? 0 : lastSelectedNode.getId());
    // 定义所属相册的编号
    String albumName = getAlbumName(fatherId, "添加相册", "请输入相册名称:");
    // 获得相册名称
    if (albumName != null) {
        // 输入了相册名称
        int id = dao.insertAlbum(fatherId, albumName); // 保存到数据库
        if (lastSelectedNode == null) {
            // 添加顶级相册
            DefaultMutableTreeNode root = (DefaultMutableTreeNode)
treeModel.getRoot(); // 获得相册树的根节点
            root.add(new MTreeNode(id, albumName)); // 添加相册节点
            treeModel.reload(); // 刷新树模型
        } else {
            // 添加子相册
            // 获得选中节点路径对象
            TreePath selectionPath = albumTree.getSelectionPath();
            if (albumTree.isExpanded(selectionPath)) { // 已经展开
                // 添加相册节点
                lastSelectedNode.add(new MTreeNode(id, albumName));
                treeModel.reload(lastSelectedNode); // 刷新指定节点
            } else {
                // 尚未展开
                albumTree.expandPath(selectionPath); // 则展开该节点
            }
        }
    }
}
```

### 3.3.3 修改相册

单击工具栏菜单的第一个“修改”按钮可以完成修改相册的功能, 该功能是通过 updAlbumButtonActionPerformed() 方法完成的, 该方法的关键代码如下:

```
private void updAlbumButtonActionPerformed(java.awt.event.ActionEvent evt)
throws UnsupportedOperationException {
    // 获得被选中的相册节点
    MTreeNode lastSelectedNode = (MTreeNode)
albumTree.getLastSelectedPathComponent();
    if (lastSelectedNode == null) {
        // 未选中任何相册
        JOptionPane.showMessageDialog(this, "请选择要修改的相册!", "友情提示",
JOptionPane.INFORMATION_MESSAGE); // 提示选中要修改的相册
    } else {
        // 存在选中的相册
        String albumName = getAlbumName(lastSelectedNode.getId(), "修改相册", "
请输入相册" +
```





```

        lastSelectedNode.getUserObject() + ""的新名称: ");
// 获得相册名称
    if (albumName != null) {
        // 将修改同步到数据库
        dao.updateAlbum(lastSelectedNode.getId(), albumName);
        File album = new File(getSelectedPath());
        album.renameTo(new
File(getSelectedPath().replace(lastSelectedNode.getUserObject().toString(),
                                albumName)));
        lastSelectedNode.setUserObject(albumName); // 修改树节点
        treeModel.reload(lastSelectedNode);        // 刷新树节点
    }
}
}

```

### 3.3.4 删除相册

单击工具栏菜单的第一个“删除”按钮可以完成删除相册的功能。该功能是通过 delAlbumButtonActionPerformed()方法完成的, 该方法的关键代码如下:

```

private void delAlbumButtonActionPerformed(java.awt.event.ActionEvent evt)
throws UnsupportedOperationException {
// 获得被选中的相册节点
MTreeNode lastSelectedNode = (MTreeNode)
albumTree.getLastSelectedPathComponent();
if (lastSelectedNode == null) {
// 未选中任何相册
JOptionPane.showMessageDialog(this, "请选择要删除的相册!", "友情提示",
JOptionPane.INFORMATION_MESSAGE); // 提示选中要删除的相册
} else {
// 存在选中的相册
int i = JOptionPane.showConfirmDialog(this, "确定要删除相册" +
lastSelectedNode.getUserObject() + ""? ",
"友情提示", JOptionPane.YES_NO_OPTION); // 询问是否真的删除
if (i == 0) {
// 确定要删除
File album = new File(getSelectedPath());
album.renameTo(new
File(getSelectedPath().replace(lastSelectedNode.getUserObject().toString(),
                                "dddddddddddddddddd")));
dao.deleteAlbum(lastSelectedNode.getId()); // 从数据库中删除
// 获得父节点
DefaultMutableTreeNode parentNode = (DefaultMutableTreeNode)
lastSelectedNode.getParent();
parentNode.remove(lastSelectedNode); // 移除该节点
treeModel.reload(parentNode);        // 刷新父节点
}
}
}
}

```

## 3.4 照片管理



### 3.4.1 功能概述

78



照片管理主要涉及添加照片、修改照片信息、删除照片、查询照片等, 用户可以使用工具栏中的按钮完成这些操作, 如图 3.13 所示。





图 3.13 照片管理相关按钮

### 3.4.2 添加照片

在添加照片时,可以采用单张添加方式,也可以采用批量添加方式。对于前一种方式,需要直接选择要添加的照片;对于后一种方式,需要选择直接包含添加照片的文件夹。添加照片的文件选择器如图 3.14 所示。

添加照片功能是在 `addPhotoButtonActionPerformed()` 方法中完成的,为了提高批量添加照片的效率,采用了线程技术,该方法的关键代码如下:



图 3.14 添加照片的文件选择器

```
private void addPhotoButtonActionPerformed(java.awt.event.ActionEvent evt)
throws UnsupportedOperationException {
    MTreeNode selectedNode = (MTreeNode)
albumTree.getLastSelectedPathComponent(); // 获得选中的节点
    if (selectedNode == null) { // 未选中任何相册
        JOptionPane.showMessageDialog(this, "请选择要添加照片的相册!", "友情提示",
JOptionPane.INFORMATION_MESSAGE); // 弹出选择相册提示
    } else { // 存在选中的相册
        JFileChooser fileChooser = new JFileChooser(); // 创建文件选择器对象
        // 设置选择模式
        fileChooser.setFileSelectionMode(JFileChooser.FILES AND DIRECTORIES);
        fileChooser.setFileFilter(swingFileFilter); // 设置文件过滤器
        int sign = fileChooser.showOpenDialog(this); // 弹出文件选择框
        if (sign == JFileChooser.APPROVE_OPTION) { // 选择了文件
            photoInfo[1] = selectedNode.getId();
            final String uploadPath = getSelectedPath();
            // 获得选择的文件对象
            File selectedFile = fileChooser.getSelectedFile();
            if (selectedFile.isDirectory()) { // 选择的为文件夹
                final File[] files = selectedFile.listFiles(ioFileFilter);
                new Thread() { // 创建并开启一个线程
                    @Override
                    public void run() { // 重构该方法
                        for (int i = 0; i < files.length; i++) { // 遍历图片文件数组
                            addPhoto(files[i], uploadPath); // 添加图片
                            try {
                                Thread.sleep(300); // 休眠 300 毫秒
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }.start();
            } else { // 选择的为图片
                addPhoto(selectedFile, uploadPath); // 添加图片
            }
        }
    }
}
```





### 3.4.3 修改照片信息

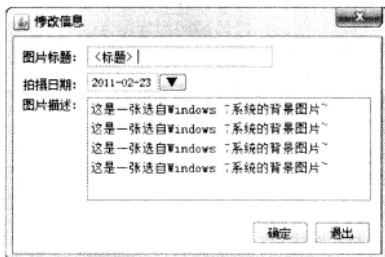


图 3.15 “修改信息”对话框

在选中一张照片之后，单击“修改”按钮可以打开“修改信息”对话框，如图 3.15 所示。照片信息包括图片标题、拍摄日期、图片描述等内容。

修改照片信息是在 `updPhotoButtonActionPerformed()` 方法中完成的，它负责处理修改照片信息按钮的动作事件。首先判断是否仅有一张照片被选中，只有满足这个条件才弹出用来修改照片信息的对话框，否则将弹出相应的信息提示框，该方法的关键代码如下：

```
private void updPhotoButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    Vector<PhotoPreviewButton> selectedPhoto =  
    PhotoPreviewButton.getSelectedPhoto(); // 获得选中的照片  
    if (selectedPhoto.size() != 1) {  
        // 每次只能修改一张照片  
        // 定义提示信息  
        String message = selectedPhoto.isEmpty() ? "请选择要修改的图片！" : "每次  
只能修改一张图片！";  
        // 弹出提示框  
        JOptionPane.showMessageDialog(this, message, "友情提示",  
JOptionPane.INFORMATION_MESSAGE);  
    } else {  
        new UpdatePhotoInfoDialog(null, true,  
selectedPhoto.firstElement()).setVisible(true); // 显示修改信息对话框  
    }  
}
```

照片标题最长只能为 20 字节，即 10 个中文汉字或 20 个英文字符。通过捕获文本框的键盘事件，可以控制文本框可输入字符的最大长度，其关键代码如下：

```
private void titleTextFieldKeyTyped(java.awt.event.KeyEvent evt) {  
    // 照片标题最多为 10 个字符  
    if (titleTextField.getText().getBytes().length == 20) {  
        evt.consume(); // 销毁此次键盘事件  
    }  
}
```

照片标题不允许为空，或者全部由空格组成，否则“确定”按钮将变为不可用，这一功能是通过为标题文本框添加 `CaretListener` 事件监听器实现的，其关键代码如下：

```
titleTextField.addCaretListener(new javax.swing.event.CaretListener() {  
    @Override  
    public void caretUpdate(javax.swing.event.CaretEvent evt) {  
        titleTextFieldCaretUpdate(evt);  
    }  
});
```

方法 `titleTextFieldCaretUpdate()` 负责处理标题文本框的 `CaretListener` 事件，其关键代码如下：

```
private void titleTextFieldCaretUpdate(javax.swing.event.CaretEvent evt) {  
    if (titleTextField.getText().trim().length() == 0) { // 标题为空  
        submitButton.setEnabled(false); // 将“确定”按钮设置为不可用  
    } else { // 标题不为空  
        if (!submitButton.isEnabled()) { // 如果“确定”按钮不可用
```







```

        submitButton.setEnabled(true);           // 将“确定”按钮设置为可用
    }
}

```

在修改完成后,单击“确定”按钮,将调用 `submitButtonActionPerformed()`方法,它负责完成修改信息的保存,该方法的关键代码如下:

```

private void submitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();                               // 销毁对话框
    selectedPhoto.setText(titleTextField.getText()); // 修改照片标题
    // 获得照片信息
    Vector<String> photoV = dao.selectPhoto(selectedPhoto.getName());
    photoV.set(2, dateTextField.getText());        // 修改日期
    photoV.set(3, titleTextField.getText());        // 修改标题
    photoV.set(4, remarkTextArea.getText());        // 修改说明
    ToolTip.set(selectedPhoto, photoV);            // 修改工具提示
    dao.updatePhoto(selectedPhoto.getName(), titleTextField.getText(),
        dateTextField.getText(),
        remarkTextArea.getText());                 // 将修改保存到数据库
}

```

### 3.4.4 删除照片

对于不需要保存的照片,用户可以将其中选中后删除。在删除前,会显示确认对话框,防止用户错误操作,如图 3.16 所示。用户也可以同时选中多张照片,然后实现批量删除。

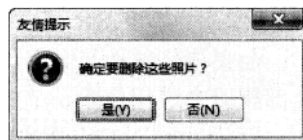


图 3.16 确认删除照片对话框

方法 `delPhotoButtonActionPerformed()`负责处理删除照片事件,其关键代码如下:

```

private void delPhotoButtonActionPerformed(java.awt.event.ActionEvent evt) {
    Vector<PhotoPreviewButton> selectedPhoto =
    PhotoPreviewButton.getSelectedPhoto(); // 获得被选中的照片
    if (selectedPhoto.isEmpty()) {        // 尚未选中任何照片
        JOptionPane.showMessageDialog(this, "请选择要删除的图片!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE); // 弹出提示
    } else {
        int i = JOptionPane.showConfirmDialog(this, "确定要删除这些照片?",
            "友情提示",
            JOptionPane.YES_NO_OPTION);    // 确认删除
        if (i == 0) {                      // 确定删除
            MPanel panel = (MPanel) photoPanel.getComponent(0); // 获得浏览方式面板
            JPanel photoBox = panel.getPhotoBoxPanel();          // 获得图片箱
            for (int j = 0; j < selectedPhoto.size(); j++) { // 遍历被选中的图片
                // 获得被选中的图片按钮
                PhotoPreviewButton button = selectedPhoto.get(j);
                photoBox.remove(button); // 从图片箱中移除
                dao.deletePhoto(button.getName()); // 从数据库中删除
                new File(button.getPath()).delete(); // 删除图片文件
            }
            SwingUtilities.updateComponentTreeUI(photoPanel); // 刷新面板
            selectedPhoto.clear(); // 清空被选中的照片
        }
    }
}

```



**注意** 在删除照片时,需要从3个位置进行删除,分别为系统窗体、数据库和照片文件。





### 3.4.5 搜索照片

如果程序中保存了大量的照片，那么提供搜索功能就变得非常重要了。单击“搜索”按钮，打开如图 3.17 所示的对话框，在该对话框中，用户可以输入各种查询条件，例如根据图片标题查询、根据拍摄时间查询、根据图片描述查询等。



图 3.17 “搜索照片”对话框

如果已经选中了某个相册，再单击“搜索”按钮，则仅在该相册中进行搜索；否则在全部相册中进行搜索。单击搜索到的照片可以看到它属于哪个相册。

方法 `findPhotoButtonActionPerformed()` 负责处理工具栏中查找照片事件，其关键代码如下：

```
private void findPhotoButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    MTreeNode selectedNode = (MTreeNode)  
albumTree.getLastSelectedPathComponent(); // 获得当前选中的节点  
    new FindPhotoInfoDialog(null, true, (selectedNode == null ? 0 :  
selectedNode.getId())) // 传入当前选中节点的主键  
    }.setVisible(true);  
}
```

输入查询条件后单击“确定”按钮，将执行 `submitButtonActionPerformed()` 方法。该方法首先获得查询条件，并判断用户是否输入了查询条件。要求至少输入一个查询条件，负责该部分业务逻辑的代码如下：

```
String title = titleTextField.getText().trim(); // 获得标题关键字  
String appointDate = appointTextField.getText(); // 获得指定日期  
String spaceStartDate = startTextField.getText(); // 获得开始日期  
String spaceEndDate = endTextField.getText(); // 获得结束日期  
String remark = remarkTextArea.getText().trim(); // 获得描述关键字  
// 验证是否填写了查询条件，要求至少填写一个查询条件  
if (title.length() == 0 && remark.length() == 0) {  
    if (appointRadioButton.isSelected()) { // 选中指定日期  
        if (appointDate.length() == 0) { // 未填写日期  
            showMessage("友情提示", "至少填写一项查找条件!"); // 弹出提示信息  
            return;  
        }  
    } else { // 选中一段时间  
        // 日期填写不完整  
        if (spaceStartDate.length() == 0 || spaceEndDate.length() == 0) {  
            // 未填写日期  
            if (spaceStartDate.length() == 0 && spaceEndDate.length() == 0) {  
                showMessage("友情提示", "至少填写一项查找条件!"); // 弹出提示信息  
            } else {  
                // 日期填写不完整
```







```

        showMessage("友情提示", "请将起止日期填写完整!"); // 弹出提示信息
    }
    return;
}
}
}

```

然后是组织查询条件并执行查询,如果拍摄日期选择的查询条件是一段期间,要求终止日期不能小于起始日期,负责该部分业务逻辑的代码如下:

```

final Vector<Vector> photos = new Vector<Vector>();
if (appointRadioButton.isSelected()) { // 选中指定日期
    char compare;
    switch (comparisonComboBox.getSelectedIndex()) {
        case 1:
            compare = '>';
            break;
        case 2:
            compare = '<';
            break;
        default:
            compare = '=';
    }
    photos.addAll(dao.selectPhoto(albumId, title, appointDate, compare,
    remark));
} else { // 选中一段时间
    DateFormat dateFormat = DateFormat.getDateInstance();
    Date startDate = null;
    Date endDate = null;
    try {
        startDate = dateFormat.parse(spaceStartDate);
        endDate = dateFormat.parse(spaceEndDate);
    } catch (ParseException e1) {
        showMessage("友情提示", "请将起止日期填写完整!"); // 弹出提示信息
        return;
    }
    if (endDate.before(startDate)) {
        showMessage("友情提示", "终止日期不能小于起始日期!"); // 弹出提示信息
        return;
    } else {
        photos.addAll(dao.selectPhoto(albumId, title, spaceStartDate,
        spaceEndDate, remark));
    }
}
}

```

最后是创建一个线程,在该线程中将符合查询条件的照片加载到当前的浏览方式面板中,负责该部分业务逻辑的代码如下:

```

final MPanel panel = (MPanel) AlbumPanel.getPhotoPanel().getComponent(0);
// 获得浏览方式面板
final JPanel photoBoxPanel = panel.getPhotoBoxPanel(); // 获得图片箱面板
photoBoxPanel.removeAll(); // 移除图片箱中的所有照片
panel.validate(); // 刷新浏览方式面板
this.dispose(); // 销毁查找照片对话框
if (photos.size() == 0) { // 没有符合条件的照片
    showMessage("查询完毕", "没有符合条件的照片!");
} else { // 存在符合条件的照片
    new Thread() { // 创建并开启一个线程
        @Override
        public void run() { // 重构该方法
            // 获得存放照片的根路径
            final String filePath = FindPhotoInfoDialog.
            class.getResource("/img/album").getPath();
            for (Vector photo : photos) { // 遍历加载符合条件的照片
                String albumPath = photo.get(0).toString(); // 相册路径
                int albumId = (Integer) photo.get(1); // 相册主键
            }
        }
    }
}

```







```
while (true) { // 向上遍历相册至根相册
    Vector album = dao.selectAlbum(albumId); // 获得父相册对象
    albumId = (Integer) album.get(1); // 获得相册主键
    albumPath = album.get(2) + "/" + albumPath; // 追加相册路径
    if (albumId == 0) { // 已遍历至根相册
        break; // 跳出循环
    }
    photoBoxPanel.add(new PhotoPreviewButton(new File(filePath + "/"
+ albumPath)));
    panel.validate(); // 刷新浏览方式面板
    try {
        Thread.sleep(600); // 休眠 600 毫秒
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}.start();
// 弹出查询完毕的提示
showMessage("查询完毕", "共有 " + photos.size() + " 张符合条件的照片!");
}
```

当单击查询得到的照片时，该照片所属的相册将被选中，这个功能是在按钮的鼠标事件监听器中完成的，负责该部分业务逻辑的代码如下：

```
String photoPath = PhotoPreviewButton.this.getPath(); // 获得照片的存放路径
photoPath = photoPath.replace('\\', '/'); // 替换字符
// 截取相册路径
photoPath = photoPath.substring(photoPath.indexOf("/img/album/") + 11);
String[] nodes = photoPath.split("/"); // 分割相册路径
JTree albumTree = AlbumPanel.getAlbumTree(); // 获得相册树对象
// 获得相册树的根节点对象
MTreeNode node = (MTreeNode) albumTree.getModel().getRoot();
for (int i = 0; i < nodes.length - 1; i++) { // 遍历相册路径
    Enumeration enu = node.children(); // 获得子节点的枚举对象
    while (enu.hasMoreElements()) { // 遍历枚举对象
        node = (MTreeNode) enu.nextElement(); // 获得节点对象
        if (node.getUserObject().equals(nodes[i])) { // 如果节点标签等于相册名称
            if (!node.isLoad()) { // 如果该节点尚未加载
                AlbumPanel.loadChildNode(node); // 加载该节点
            }
            break; // 跳出循环
        }
    }
}
TreePath treePath = new TreePath(node.getPath()); // 创建相册的路径对象
albumTree.scrollPathToVisible(treePath); // 滚动该节点至可见
TreeSelectionListener treeSelectionListener = albumTree.
getTreeSelectionListeners()[0]; // 获得树的选中事件监听器
// 移除选中事件监听器
albumTree.removeTreeSelectionListener(treeSelectionListener);
albumTree.setSelectionPath(treePath); // 选中该路径节点
albumTree.addTreeSelectionListener(treeSelectionListener); // 添加选中事件监听器
```



说明 当相册树足够大时，可能有部分相册节点不可见，通过 JTree 类的 scrollPathToVisible

(TreePath path)方法，可以使路径 path 中的叶子节点滚动至可见。





### 3.4.6 保存图片

本模块不仅能够将照片保存到数据库中，还能够将数据库中保存的照片复制出来。在选中一张或多张照片后，单击“保存”按钮，就可以将它（或它们）保存到指定的位置。保存图片的对话框如图 3.18 所示。

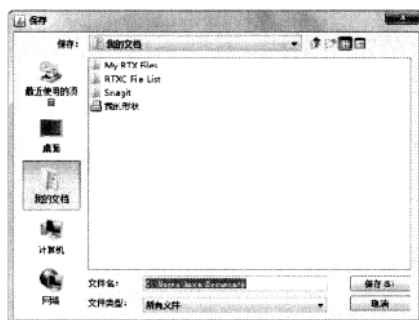


图 3.18 “保存”对话框

保存图片的功能是通过 `saveButtonActionPerformed()` 方法实现的，该方法的关键代码如下：

```
private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 获得选中的照片
    Vector<PhotoPreviewButton> photos = PhotoPreviewButton.getSelectedPhoto();
    int amount = photos.size(); // 获得选中照片数量
    if (amount == 0) { // 尚未选中任何照片
        JOptionPane.showMessageDialog(null, "请先选择要保存的照片！", "友情提示",
            JOptionPane.INFORMATION_MESSAGE); // 弹出提示信息
    } else {
        JFileChooser pathChooser = new JFileChooser(); // 创建文件选择框对象
        // 只允许选择文件夹
        pathChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        // 弹出对话框并获得操作标记值
        int optionSign = pathChooser.showSaveDialog(this);
        if (optionSign == JFileChooser.APPROVE_OPTION) { // 如果选择了文件夹
            // 获得选择的文件夹对象
            File selectedPath = pathChooser.getSelectedFile();
            // 获得最后一个照片对象
            PhotoPreviewButton lastPhoto = photos.lastElement();
            if (lastPhoto == photos.get(0)) { // 和第一个照片重复
                photos.remove(amount - 1); // 移除最后一个照片
            } else {
                if (lastPhoto == photos.get(amount - 2)) { // 和倒数第二个照片重复
                    photos.remove(amount - 1); // 移除最后一个照片
                }
            }
        }
        int num = 1; // 照片编号
        for (PhotoPreviewButton photo : photos) { // 遍历照片
            try {
                // 创建文件输入流对象
                InputStream inStream = new FileInputStream(photo.getPath());
                // 定义照片名称
                String name = (num++) + "." + photo.getText() +
                    photo.getName().substring(23);
                // 创建文件输出流对象
```





```
        OutputStream outStream = new
        FileOutputStream(selectedPath.getPath() + "/" + name);
        int readBytes = 0; // 读取字节数
        byte[] buffer = new byte[10240]; // 定义缓存数组
        // 从输入流读取数据到缓存数组中
        while ((readBytes = inStream.read(buffer, 0, 10240)) != -1) {
            // 输出缓存数组中的数据到输出流
            outStream.write(buffer, 0, readBytes);
        }
        outStream.close(); // 关闭输出流对象
        inStream.close(); // 关闭输入流对象
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
}
```

## 3.5 照片显示

### 3.5.1 功能概述

在缩略图和幻灯片方式下，双击照片可以使其全屏显示，在该状态下双击鼠标或按【Esc】键可以退出全屏显示。全屏显示照片的效果如图 3.19 所示。

此外，还可以使用播放器方式浏览照片。在播放器方式中，提供了两种模式：自动模式和手动模式。使用自动模式，可以让照片在指定的间隔时间后自动切换。使用手动模式，可以用按钮来切换照片。使用手动模式的运行效果如图 3.20 所示。

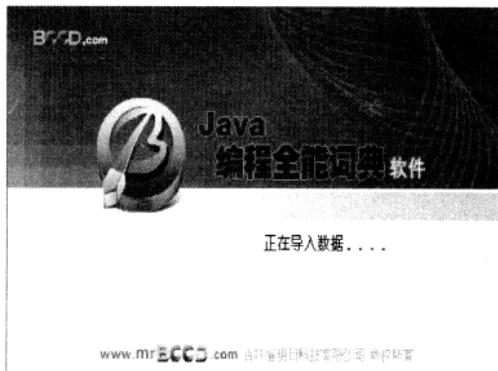


图 3.19 全屏显示照片

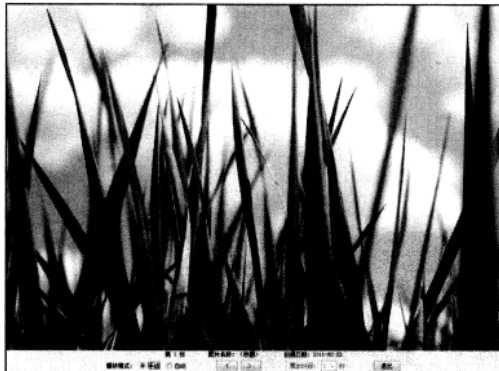


图 3.20 手动模式播放照片



### 3.5.2 全屏显示照片

捕获双击照片事件是通过鼠标事件监听器实现的，全屏显示照片功能是通过 ShowDialog 类实现的。所以在捕获到非单击事件后，就是创建该类的对话框，并令其可



见。用来捕获鼠标双击事件的关键代码如下：

```
addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 1) { // 单击
            // 省略部分代码
        } else { // 连击
            new ShowDialog(photoFile).setVisible(true); // 全屏显示照片
        }
    }
});
```

为了能够真正的全屏显示照片，需要将对话框窗体设置成不显示标题栏，并将绘制范围设置为整个屏幕，其关键代码如下：

```
setModal(true); // 设置对话框为有模式
setUndecorated(true); // 设置不显示对话框的标题栏
// 设置窗体的绘制范围
setBounds(0, 0, ScreenSize.getWidth(), ScreenSize.getHeight());
```

双击鼠标可以退出全屏显示，也是通过鼠标事件监听器实现的。当捕获到双击鼠标的事件时，就销毁对话框，其关键代码如下：

```
photoLabel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) { // 如果是双击鼠标
            ShowDialog.this.dispose(); // 则销毁对话框窗体
        }
    }
});
```

按【Esc】键可以退出全屏显示，这是通过添加键盘事件监听器实现的，当捕获到【Esc】键被按下时，就销毁对话框，其关键代码如下：

```
addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ESCAPE) { // 如果〈Esc〉键被按下
            ShowDialog.this.dispose(); // 则销毁对话框窗体
        }
    }
});
```

### 3.5.3 照片播放器

方法 `handRadioButtonActionPerformed()` 负责处理“手动”单选按钮事件，包括设置当前的播放模式和控件的可用性，其关键代码如下：

```
private void handRadioButtonActionPerformed(java.awt.event.ActionEvent evt) {
    isPlay = false; // 采用手动查看模式
    previousButton.setEnabled(true); // 设置“上一张”按钮可用
    nextButton.setEnabled(true); // 设置“下一张”按钮可用
    timeLabel.setEnabled(false); // 设置展示时间标签不可用
    secondComboBox.setEnabled(false); // 设置时间组合框不可用
    secondLabel.setEnabled(false); // 设置单位标签不可用
}
```

方法 `autoRadioButtonActionPerformed()` 负责处理“自动”单选按钮事件，包括设置当前的播放模式和控件的可用性。关键是创建一个线程实现图片的自动播放，并随时读取每





张照片的播放时间，以及是否暂停了播放，其关键代码如下：

```
private void autoRadioButtonActionPerformed(java.awt.event.ActionEvent evt) {
    isPlay = true; // 采用自动播放模式
    previousButton.setEnabled(false); // 设置“上一张”按钮不可用
    nextButton.setEnabled(false); // 设置“下一张”按钮不可用
    timeLabel.setEnabled(true); // 设置展示时间标签可用
    secondComboBox.setEnabled(true); // 设置时间组合框可用
    secondLabel.setEnabled(true); // 设置单位标签可用
    new Thread() {
        @Override
        public void run() { // 重构该方法
            close: while (isPlay) { // 如果采用自动播放模式
                for (int i = showIndex; i < maxIndex + 1; i++) { // 遍历照片
                    showPointedPhoto(i); // 显示指定照片
                    try {
                        Thread.sleep((secondComboBox.getSelectedIndex() + 1) *
1000); // 休眠指定秒
                        while (isPause) { // 如果暂停播放
                            Thread.sleep(500); // 休眠 500 毫秒
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    if (!isPlay) { // 如果采用手动查看模式
                        break close; // 停止播放
                    }
                }
                int i = JOptionPane.showOptionDialog(PlayDialog.this,
"图片播放完毕!", "播放完毕",
JOptionPane.INFORMATION_MESSAGE, JOptionPane.YES_OPTION,
null, new String[] { "重复播放", "从头播放", "退出播放" },
"退出播放");
                if (i > 0) {
                    if (i == 1) { // 从头播放
                        showIndex = 0;
                    } else { // 退出播放
                        isPlay = false; // 停止播放
                        PlayDialog.this.dispose(); // 销毁播放器对话框
                    }
                }
            }
        }
    }.start();
}
```

方法 `previousButtonActionPerformed()` 负责处理“上一张”按钮事件，包括更换当前显示的照片和判断“上一张”按钮和“下一张”按钮是否可用，其关键代码如下：

```
private void previousButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (showIndex == maxIndex) { // 如果显示照片的索引等于最后一张照片的索引
        nextButton.setEnabled(true); // 设置“下一张”按钮可用
    }
    showPointedPhoto(--showIndex); // 显示指定照片
    if (showIndex == 0) { // 如果显示照片的索引等于 0
        previousButton.setEnabled(false); // 设置“上一张”按钮不可用
    }
}
```

方法 `nextButtonActionPerformed()` 负责处理“下一张”按钮事件，包括更换当前显示的照片和判断“上一张”按钮和“下一张”按钮是否可用，其关键代码如下：

```
private void nextButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (showIndex == 0) { // 如果显示照片的索引等于 0
        previousButton.setEnabled(true); // 设置“上一张”按钮可用
    }
}
```







```
showPointedPhoto(++showIndex);    // 显示指定照片
if (showIndex == maxIndex) {       // 如果显示照片的索引等于最后一张照片的索引
    nextButton.setEnabled(false);  // 设置“下一张”按钮不可用
}
```

当光标移动到照片上时，将触发 `photoLabelMouseEntered()` 方法，它负责处理鼠标移入事件。在该方法中仅用来标记是否暂停播放的 `isPause` 属性设为 `true`，其关键代码如下：

```
private void photoLabelMouseEntered(java.awt.event.MouseEvent evt) {
    isPause = true;                // 暂停播放
}
```

当光标离开照片时，将触发 `photoLabelMouseExited()` 方法，它负责处理鼠标移出事件。在该方法中仅将用来标记是否暂停播放的 `isPause` 属性设为 `false`，其关键代码如下：

```
private void photoLabelMouseExited(java.awt.event.MouseEvent evt) {
    isPause = false;              // 继续播放
}
```

在退出播放器浏览方式时，不仅要销毁播放器对话框，还要将用来标记是否播放的 `isPlay` 属性设为 `false`，否则可能因为在播放的过程中退出，而没有真正停止用来播放的线程。负责处理“退出”按钮事件的 `exitButtonActionPerformed()` 方法的关键代码如下：

```
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    isPlay = false;               // 停止播放
    dispose();                   // 销毁播放器对话框
}
```



# 第 4 章

---

## 定制打印模块

( Swing+MySQL 实现 )

随着社会的发展，人们的生活节奏不断加快，快递行业已经成为快速发展的行业。在快递过程中，需要填写大量的表单，例如物品信息等。为了提高快递的效率，可以采用计算机来辅助表单的填写工作。本章将开发一个定制打印模块，该系统支持表单内容的记录与打印。主要使用 PrintJob 类获得打印对象与实现打印，通过实现 Printable 接口中的 print() 方法设置打印内容的位置。通过本章的学习，读者可以学习以下内容：

- » 获取打印对象
- » 设置打印内容
- » 添加与修改快递信息
- » 打印和设置快递信息





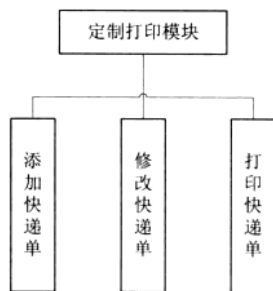
## 4.1 定制打印模块概述

### 4.1.1 模块概述

随着社会的发展,人们的生活节奏不断加快。为了节约宝贵的时间,快递业务应运而生。在快递过程中,需要填写大量的表单,如果使用计算机来辅助填写及保存相应的记录,则能大大提高快递的效率。因此,需要开发一个定制打印模块,该模块应该支持快速录入关键信息,例如发件人和收件人的姓名、电话和地址等,快递物品的信息等,并将其保存在数据库中以便以后查看。读者可以在本模块的基础上开发快递打印系统。

### 4.1.2 功能结构

定制打印模块中主要包括添加快递单、修改快递单和打印快递单 3 个子模块,其功能结构如图 4.1 所示。



### 4.1.3 程序预览

定制打印模块由多个窗体组成,主窗体的运行效果如图 4.2 所示,主要功能是调用执行本系统的所有功能。

图 4.1 定制打印模块功能结构图



图 4.2 模块主窗体

“添加快递单”窗体的运行效果如图 4.3 所示,主要功能是完成快递单的编辑工作,这些信息包括发件人姓名、电话、地址和收件人姓名、电话、地址等。





图 4.3 “添加快递单”窗体

“修改快递单”窗体的运行效果如图 4.4 所示，主要功能是完成对已经保存的快递信息的修改操作。

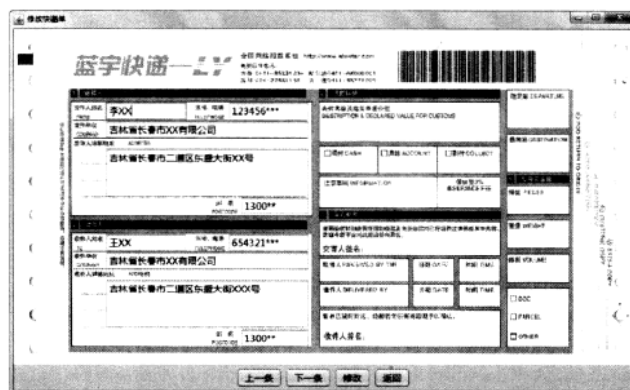


图 4.4 “修改快递单”窗体

“打印快递单”窗体的运行效果如图 4.5 所示，主要功能是完成快递单的打印。



图 4.5 “打印快递单”窗体





## 4.2 关键技术

### 4.2.1 自定义面板背景图片

在 Swing 中, 默认提供的控件并不美观。为了让程序更加好看, 首先开发一个支持背景图片的面板类。在 com.lw.util 包中创建 BackgroundPanel 类, 它继承了 JPanel 类, 并重写了 paintComponent() 方法, 其代码如下:

```
public class BackgroundPanel extends JPanel {
    private static final long serialVersionUID = -7411632419492480055L;
    private Image image;
    public BackgroundPanel(Image image) {
        this.image = image;
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // 调用父类的方法
        if (image != null) {
            int width = getWidth(); // 获得面板的宽度
            int height = getHeight(); // 获得面板的高度
            g.drawImage(image, 0, 0, width, height, this); // 绘制图像
        }
    }
}
```

paintComponent() 方法可以用来重新绘制控件, 该方法的声明如下:

```
protected void printComponent(Graphics g)
```

g: 进行绘制的 Graphics 上下文。

为了让背景图片刚好填充整个面板, 需要获得面板的宽度和高度, 这里使用 getWidth() 和 getHeight() 两个方法, 其声明如下:

```
public int getWidth()
public int getHeight()
```

为了绘制背景图片, 使用了 Graphics 类中定义的 drawImage() 方法, 这个方法在 Graphics 类中具有多种重载形式, BackgroundPanel 类中使用形式声明如下:

```
public abstract boolean drawImage(Image img, int x, int y, int width, int height,
ImageObserver observer)
```

其中各个参数的说明如表 4.1 所示。

表 4.1 drawImage() 方法参数说明

参 数	说 明
img	需要绘制的图片对象
x	绘制的起始位置的 x 坐标
y	绘制的起始位置的 y 坐标
width	绘制区域的宽度
height	绘制区域的高度
observer	当转换更多图片时需要同时转换的对象





## 4.2.2 窗体居中显示

为了让使用者更加舒适,可以设置窗体显示位置为居中显示。这就需要获得用户显示器的大小并根据窗体大小来计算显示位置。这是通过工具类 WindowUtil 实现的,该类的代码如下:

```
public class WindowUtil {
    // 将窗体大小设置成 1000 像素×618 像素
    public static Dimension getSize() {
        return new Dimension(1000, 618);
    }
    // 计算窗体居中显示时左上角坐标
    public static Point getLocation() {
        Toolkit toolKit = Toolkit.getDefaultToolkit(); // 获得 Toolkit 实例
        Dimension screenSize = toolKit.getScreenSize(); // 获得显示器大小
        if ((screenSize.width < getSize().width) || (screenSize.height <
            getSize().height)) {
            JOptionPane.showMessageDialog(null, "显示器分辨率至少为 1000×618", "",
                JOptionPane.WARNING_MESSAGE);
            System.exit(0); // 终止程序
        }
        int x = (screenSize.width - getSize().width) / 2; // 计算左上角横坐标
        int y = (screenSize.height - getSize().height) / 2; // 计算左上角纵坐标
        return new Point(x, y);
    }
}
```

首先定义 getSize()方法,它返回窗体的大小,这里将窗体的宽度设置为 1000 像素,高度设置为 618 像素。Dimension 类用于封装控件的宽度和高度,使用该类有利于以面向对象的方式编写代码。

接着定义 getLocation()方法,它返回窗体左上角的坐标。这里首先获得 Toolkit 类的对象,由于该类是抽象类,不能直接使用 new 操作符实例化,因此使用该类定义的 getDefaultToolkit()方法获得实例。该方法的声明如下:

```
public static Toolkit getDefaultToolkit()
```

Toolkit 类中的 getScreenSize()方法可以获得用户显示器的分辨率,该方法的声明如下:

```
public abstract Dimension getScreenSize() throws HeadlessException
```

如果用户的分辨率小于 1000 像素×618 像素则弹出提示对话框并终止程序。

## 4.2.3 使用 JavaBean 封装信息

在本模块中,用户需要填写快递单,这涉及了大量的信息,例如发件人姓名、电话、地址,收件人姓名、电话、地址等,通常定义一个类来保存这些信息,将它们设置成私有字段,并提供对应的 get 和 set 方法,这样就构成了一个 JavaBean。ExpressListBean 的代码如下:

```
public class ExpressListBean {
    private int id;
    private String senderName;
    private String senderPhone;
    private String senderCompany;
    private String senderAddress;
```





```
private String senderCode;
private String receiverName;
private String receiverPhone;
private String receiverCompany;
private String receiverAddress;
private String receiverCode;
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getSenderName() {
    return senderName;
}
public void setSenderName(String senderName) {
    this.senderName = senderName;
}
public String getSenderPhone() {
    return senderPhone;
}
public void setSenderPhone(String senderPhone) {
    this.senderPhone = senderPhone;
}
public String getSenderCompany() {
    return senderCompany;
}
public void setSenderCompany(String senderCompany) {
    this.senderCompany = senderCompany;
}
public String getSenderAddress() {
    return senderAddress;
}
public void setSenderAddress(String senderAddress) {
    this.senderAddress = senderAddress;
}
public String getSenderCode() {
    return senderCode;
}
public void setSenderCode(String senderCode) {
    this.senderCode = senderCode;
}
public String getReceiverName() {
    return receiverName;
}
public void setReceiverName(String receiverName) {
    this.receiverName = receiverName;
}
public String getReceiverPhone() {
    return receiverPhone;
}
public void setReceiverPhone(String receiverPhone) {
    this.receiverPhone = receiverPhone;
}
public String getReceiverCompany() {
    return receiverCompany;
}
public void setReceiverCompany(String receiverCompany) {
    this.receiverCompany = receiverCompany;
}
public String getReceiverAddress() {
    return receiverAddress;
}
public void setReceiverAddress(String receiverAddress) {
    this.receiverAddress = receiverAddress;
}
public String getReceiverCode() {
```







```
        return receiverCode;
    }
    public void setReceiverCode(String receiverCode) {
        this.receiverCode = receiverCode;
    }
}
```

#### 4.2.4 获得 MySQL 数据库连接

本程序的后台使用 MySQL 数据库来保存快递单信息，在使用 JDBC 操作数据库时，需要先获得数据库连接对象，这是使用 JdbcHelper 类的 getConnection() 方法实现的，该方法的代码如下：

```
private static Connection getConnection() {
    Connection conn = null;
    try {
        Class.forName(DRIVER); // 加载数据库驱动
        // 获得数据库连接
        conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        return conn;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

方法中用到的 DRIVER、URL、USERNAME 和 PASSWORD 是字符串常量，定义在 JdbcConfig 接口中，该接口的代码如下：

```
public interface JdbcConfig {
    String DRIVER = "com.mysql.jdbc.Driver";
    String URL = "jdbc:mysql://localhost:3306/db_ExpressPrintModule";
    String USERNAME = "root";
    String PASSWORD = "111";
}
```



在使用 JDBC 操作数据库时需要为项目添加数据库驱动文件。

#### 4.2.5 批量处理数据库操作

在处理大量数据时，使用批处理操作能够大幅度提高效率，这里使用该技术完成快递单的保存和修改操作。

##### 1. 保存快递单信息

使用 JdbcHelper 类的 save() 方法来保存快递单信息，其关键代码如下：

```
public static int save(ExpressListBean expressList) {
    // 定义保存快递单信息的批处理语句
    String sql = "insert into tb_expressList (senderName, senderPhone, senderCompany, senderAddress, senderCode, receiverName, receiverPhone, receiverCompany, receiverAddress, receiverCode) values";
}
```







```
(?, ?, ?, ?, ?, ?, ?, ?, ?, ?);";
Connection conn = getConnection();
PreparedStatement ps = null;
try {
    // 保存数据
    ps = getConnection().prepareStatement(sql);
    ps.setString(1, expressList.getSenderName());
    ps.setString(2, expressList.getSenderPhone());
    ps.setString(3, expressList.getSenderCompany());
    ps.setString(4, expressList.getSenderAddress());
    ps.setString(5, expressList.getSenderCode());
    ps.setString(6, expressList.getReceiverName());
    ps.setString(7, expressList.getReceiverPhone());
    ps.setString(8, expressList.getReceiverCompany());
    ps.setString(9, expressList.getReceiverAddress());
    ps.setString(10, expressList.getReceiverCode());
    return ps.executeUpdate(); // 获得更新数据的行数
} catch (SQLException e) {
    e.printStackTrace();
} finally { // 释放资源
    if (ps != null) {
        try {
            ps.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
return -1; // 如果发生异常则返回-1 作为标识
}
```

使用 PreparedStatement 接口可以完成批处理操作。批处理时使用问号来表示需要添加的变量，然后使用该接口中与变量类型对应的 set() 方法来添加变量。例如 setString() 表示设置字符串的，这个方法第一个参数表示变量的位置，第二个参数表示变量的值。最后调用 executeUpdate() 方法来完成批量保存。

## 2. 修改快递单

修改快递单使用的是 JdbcHelper 类中的 update() 方法，其关键代码如下：

```
public static int update(ExpressListBean expressList) {
    // 定义修改快递单信息的批处理语句
    String sql = "update tb_expressList set senderName = ?, senderPhone = ?,
senderCompany = ?,
senderAddress = ?, senderCode = ?, receiverName = ?,
receiverPhone = ?,
receiverCompany = ?, receiverAddress = ?, receiverCode = ? where
id = ?";
    Connection conn = getConnection();
    PreparedStatement ps = null;
    try {
        // 保存数据
        ps = getConnection().prepareStatement(sql);
        ps.setString(1, expressList.getSenderName());
        ps.setString(2, expressList.getSenderPhone());
        ps.setString(3, expressList.getSenderCompany());
        ps.setString(4, expressList.getSenderAddress());
        ps.setString(5, expressList.getSenderCode());
        ps.setString(6, expressList.getReceiverName());
        ps.setString(7, expressList.getReceiverPhone());
    }
```



97







```
        ps.setString(8, expressList.getReceiverCompany());
        ps.setString(9, expressList.getReceiverAddress());
        ps.setString(10, expressList.getReceiverCode());
        ps.setInt(11, expressList.getId());
        return ps.executeUpdate(); // 获得更新数据的行数
    } catch (SQLException e) {
        e.printStackTrace();
    } finally { // 释放资源
        if (ps != null) {
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return -1; // 如果发生异常则返回-1 作为标识
}
```

#### 4.2.6 使用 List 保存查询结果

在 JDBC 中，查询操作的返回值是 `ResultSet` 类型的对象，它使用起来并不方便，因此通常将结果保存到 `List` 中。本模块使用 `JdbcHelper` 类的 `query()` 方法来保存表格中的全部数据，其关键代码如下：

```
public static List<ExpressListBean> query() {
    String sql = "select * from tb_expressList;"; // 定义查询语句
    List<ExpressListBean> results = new ArrayList<ExpressListBean>();
    Connection conn = getConnection();
    Statement stat = null;
    ResultSet rs = null;
    try {
        stat = conn.createStatement();
        rs = stat.executeQuery(sql);
        while (rs.next()) {
            ExpressListBean expressList = new ExpressListBean();
            expressList.setId(rs.getInt("id"));
            expressList.setSenderName(rs.getString("senderName"));
            expressList.setSenderPhone(rs.getString("senderPhone"));
            expressList.setSenderCompany(rs.getString("senderCompany"));
            expressList.setSenderAddress(rs.getString("senderAddress"));
            expressList.setSenderCode(rs.getString("senderCode"));
            expressList.setReceiverName(rs.getString("receiverName"));
            expressList.setReceiverPhone(rs.getString("receiverPhone"));
            expressList.setReceiverCompany(rs.getString("receiverCompany"));
            expressList.setReceiverAddress(rs.getString("receiverAddress"));
            expressList.setReceiverCode(rs.getString("receiverCode"));
            results.add(expressList);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally { // 释放资源
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```







```

        if (stat != null) {
            try {
                stat.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    return results;
}

```



**注意** 如果数据表中有大量数据, 这种处理方式效率是非常低的, 推荐增加查询条件来显示结果数量。

## 4.2.7 使用 Java 操作打印机

打印是本模块的核心功能, 使用 Java 来完成打印功能, 主要步骤如下:

(1) 获得 PrinterJob 对象。

PrinterJob 是控制打印的主要类, 应用程序调用该类的方法来建立打印任务, 可以选择是否为用户激活打印对话框, 然后打印文件。由于该类是抽象类, 不能够使用 new 操作符进行实例化, 这里使用 getPrinterJob() 方法来获得该类实例, 该方法的声明如下:

```
public static PrinterJob getPrinterJob()
```

(2) 启动打印对话框。

printDialog() 方法将打开一个本地对话框, 用来给用户修改打印属性, 这些属性包括纸质大小、打印方向等, 其运行效果如图 4.6 所示。

单击图 4.6 中的“属性”按钮, 弹出的对话框如图 4.7 所示。

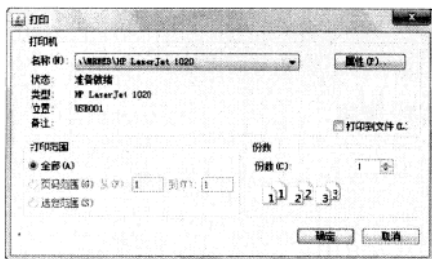


图 4.6 打印对话框

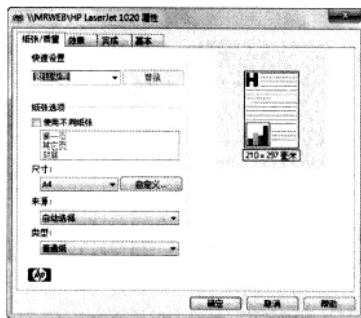


图 4.7 “打印属性”对话框

在图 4.7 中, 用户可以配置纸张的尺寸、类型等属性。





(3) 绘制打印内容。

在完成打印机配置之后，还需要设置打印内容，这是使用 `setPrintable()` 方法完成的，该方法的声明如下：

```
public abstract void setPrintable(Printable painter)
```

painter: 包含打印内容的 Printable 对象。

在 Printable 接口中，定义了一个 `print()` 方法，该方法的声明如下：

```
int print(Graphics graphics, PageFormat pageFormat, int pageIndex) throws PrinterException
```

- graphics: 绘制打印信息的上下文。
- pageFormat: 包括页面大小和方向信息的 PageFormat 对象。
- pageIndex: 页面的页码，从 0 开始计数。

(4) 开始打印。

如果 `print()` 方法的返回值是 `Printable.PAGE_EXISTS`，则表示准备工作已经完成，可以开始打印。

## 4.3 主窗体

### 4.3.1 功能概述

定制打印模块主界面简洁美观，通过主窗体可以完成模块的全部操作。包括添加快递单信息、修改快递单信息、打印快递单等。定制打印模块主界面的运行效果如图 4.8 所示。

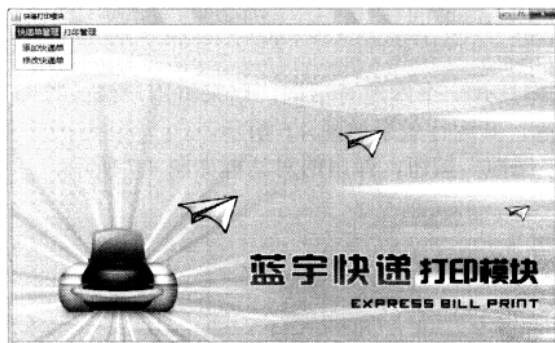


图 4.8 模块主窗体



### 4.3.2 添加菜单及菜单项

100



在主窗体中，使用菜单来管理各个功能，下面介绍菜单及菜单项的使用。在窗体中添加菜单前，需要先添加菜单栏，即 `JMenuBar`，其关键代码如下：

```
JMenuBar menuBar = new JMenuBar();  
setJMenuBar(menuBar);
```





接着向菜单栏中增加菜单，其关键代码如下：

```
JMenu expressListManagementMenu = new JMenu("快递单管理");
expressListManagementMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15));
menuBar.add(expressListManagementMenu);

JMenu printManagementMenu = new JMenu("打印管理");
printManagementMenu.setFont(new Font("微软雅黑", Font.PLAIN, 15));
menuBar.add(printManagementMenu);
```

这里创建了“快递单管理”和“打印管理”两个菜单，下面为这两个菜单分别增加菜单项，其关键代码如下：

```
JMenuItem addExpressListMenuItem = new JMenuItem("添加快递单");
// 省略事件处理相关代码
addExpressListMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15));
expressListManagementMenu.add(addExpressListMenuItem);

JMenuItem modifyExpressListMenuItem = new JMenuItem("修改快递单");
// 省略事件处理相关代码
modifyExpressListMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15));
expressListManagementMenu.add(modifyExpressListMenuItem);

JMenuItem printExpressListMenuItem = new JMenuItem("打印快递单");
// 省略事件处理相关代码
printExpressListMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 15));
printManagementMenu.add(printExpressListMenuItem);
```

在上述代码中，统一将字体设置成微软雅黑，大小是 15。为了处理用户单击菜单项事件，可以为其增加事件监听器，下面以“打印快递单”菜单项为例进行讲解。添加事件监听器的关键代码如下：

```
printExpressListMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_printExpressListMenuItem_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_printExpressListMenuItem_actionPerformed()` 方法，它是由 IDE 工具自动生成的，用于创建 `PrintManagementFrame` 并将其设置成可见。其关键代码如下：

```
protected void do_printExpressListMenuItem_actionPerformed(ActionEvent e) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                PrintManagementFrame frame = new PrintManagementFrame();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

### 4.3.3 加载窗体背景图片

窗体中的主体部分是一张图片，这是使用自定义的 `BackgroundPanel` 类完成加载的，其关键代码如下：

```
URL resource = MainFrame.class.getResource("/image/main.jpg");
ImageIcon icon = new ImageIcon(resource);
```





```
BackgroundPanel backgroundPanel = new BackgroundPanel(icon.getImage());
getContentPane().add(backgroundPanel);
```

### 4.3.4 设置窗体显示位置和大小

在使用窗体之前，需要为其设置显示位置和大小。因为 Swing 中窗体的默认大小是  $0 \times 0$ ，这里使用自定义的 WindowUtil 类，其关键代码如下：

```
setLocation(WindowUtil.getLocation());
setSize(WindowUtil.getSize());
```



对于其他的窗体也采用类似的方式设置显示位置和大小，下面省略这部分的

讲解。

至此完成了 MainFrame 类的编写，关于该类的详细代码请读者参考源文件。

## 4.4 添加快递单

### 4.4.1 功能概述

添加快递单窗体用于添加快递信息，包括寄件人和收件人的相关信息。单击主窗体“快递单管理”/“添加快递单”菜单项，就可以打开“添加快递单”窗体，如图 4.9 所示。

图 4.9 “添加快递单”窗体





## 4.4.2 加载快递单图片

在图 4.9 中显示的快递单是一张图片, 通过自定义的 `BackgroundPanel` 类完成加载, 其关键代码如下:

```
// 获得图片资源
URL resource = ExpressAdditionFrame.class.getResource
("/image/expressList.jpg");
ImageIcon icon = new ImageIcon(resource);           // 利用图片资源创建图标
// 创建带背景图片的面板
BackgroundPanel backgroundPanel = new BackgroundPanel(icon.getImage());
contentPane.add(backgroundPanel, BorderLayout.CENTER); // 应用面板
```

## 4.4.3 设置文本框和文本域控件

快递单上的信息需要使用文本框和文本域控件进行接收。在加载完快递单图片后, 将面板布局管理器设置成 `null`, 然后使用绝对布局来摆放各个控件, 其关键代码如下:

```
senderNameTextField = new JTextField();           // 创建发件人姓名文本域
senderNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderNameTextField.setBounds(145, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderNameTextField);         // 应用文本域控件
senderNameTextField.setColumns(10);               // 设置文本域列数

senderPhoneTextField = new JTextField();           // 创建发件人电话文本域
senderPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderPhoneTextField.setBounds(340, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderPhoneTextField);        // 应用文本域控件
senderPhoneTextField.setColumns(10);              // 设置文本域列数

senderCompanyTextField = new JTextField();         // 创建发件人公司文本域
// 修改字体
senderCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
senderCompanyTextField.setBounds(145, 146, 325, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderCompanyTextField);      // 应用文本域控件
senderCompanyTextField.setColumns(10);            // 设置文本域列数

senderAddressTextArea = new JTextArea();           // 创建收件人地址文本域
senderAddressTextArea.setLineWrap(true);          // 支持文本区自动换行
senderAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderAddressTextArea.setBounds(145, 190, 325, 80); // 设置文本区控件的位置和大小
backgroundPanel.add(senderAddressTextArea);       // 应用文本区控件

senderCodeTextField = new JTextField();            // 创建发件人邮编文本域
senderCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderCodeTextField.setBounds(360, 268, 110, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderCodeTextField);         // 应用文本域控件
senderCodeTextField.setColumns(10);               // 设置文本域列数

receiverNameTextField = new JTextField();          // 创建收件人姓名文本域
receiverNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverNameTextField.setBounds(145, 327, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverNameTextField);       // 应用文本域控件
receiverNameTextField.setColumns(10);             // 设置文本域列数

receiverPhoneTextField = new JTextField();         // 创建收件人电话文本域
// 修改字体
receiverPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
receiverPhoneTextField.setBounds(340, 327, 130, 30); // 设置文本域控件的位置和大小
```





```
backgroundPanel.add(receiverPhoneTextField); // 应用文本域控件
receiverPhoneTextField.setColumns(10); // 设置文本域列数

receiverCompanyTextField = new JTextField(); // 创建收件人公司文本域
// 修改字体
receiverCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
// 设置文本域控件的位置和大小
receiverCompanyTextField.setBounds(145, 356, 325, 30);
backgroundPanel.add(receiverCompanyTextField); // 应用文本域控件
receiverCompanyTextField.setColumns(10); // 设置文本域列数
receiverAddressTextArea = new JTextArea(); // 创建收件人地址文本域
receiverAddressTextArea.setLineWrap(true); // 支持文本区自动换行
// 修改字体
receiverAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15));
// 设置文本区控件的位置和大小
receiverAddressTextArea.setBounds(145, 400, 325, 80);
backgroundPanel.add(receiverAddressTextArea); // 应用文本区控件

receiverCodeTextField = new JTextField(); // 创建收件人邮编文本域
receiverCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverCodeTextField.setBounds(360, 480, 110, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverCodeTextField); // 应用文本域控件
receiverCodeTextField.setColumns(10); // 设置文本域列数
```

### 4.4.4 添加工具按钮

在窗体中使用了“保存”、“清空”和“返回”三个按钮，并将它们放置在一个面板中，其关键的代码如下：

```
JPanel buttonPanel = new JPanel(); // 创建按钮面板
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用按钮面板

JButton saveButton = new JButton("保存"); // 创建“保存”按钮
// 省略事件处理相关代码
saveButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(saveButton); // 应用按钮

JButton clearButton = new JButton("清空"); // 创建“清空”按钮
// 省略事件处理相关代码
clearButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(clearButton); // 应用按钮

JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(returnButton); // 应用按钮
```

### 4.4.5 保存快递单信息

通过监听“保存”按钮单击事件，完成对用户添加的快递单信息的保存功能。由于快递单上的信息都非常重要，因此不能为空值。“保存”按钮事件监听器关键代码如下：

```
saveButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_saveButton_actionPerformed(e);
    }
});
```







在事件监听器中，调用了 `do_saveButton_actionPerformed()` 方法，它是 IDE 工具自动生成的方法，用于对文本域和文本区的非空校验及保存用户输入信息，其关键代码如下：

```
protected void do_saveButton_actionPerformed(ActionEvent e) {
    // 获得用户输入的信息并进行非空校验
    String senderName = senderNameTextField.getText();
    if (senderName.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人姓名!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderPhone = senderPhoneTextField.getText();
    if (senderPhone.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人电话!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderCompany = senderCompanyTextField.getText();
    if (senderCompany.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人公司!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderAddress = senderAddressTextArea.getText();
    if (senderAddress.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人地址!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderCode = senderCodeTextField.getText();
    if (senderCode.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人邮编!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverName = receiverNameTextField.getText();
    if (receiverName.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人姓名!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverPhone = receiverPhoneTextField.getText();
    if (receiverPhone.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人电话!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverCompany = receiverCompanyTextField.getText();
    if (receiverCompany.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人公司!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverAddress = receiverAddressTextArea.getText();
    if (receiverAddress.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人地址!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverCode = receiverCodeTextField.getText();
    if (receiverCode.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人邮编!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    // 将获得的信息保存到 ExpressListBean 中
}
```





```
ExpressListBean expressList = new ExpressListBean();
expressList.setSenderName(senderName);
expressList.setSenderPhone(senderPhone);
expressList.setSenderCompany(senderCompany);
expressList.setSenderAddress(senderAddress);
expressList.setSenderCode(senderCode);
expressList.setReceiverName(receiverName);
expressList.setReceiverPhone(receiverPhone);
expressList.setReceiverCompany(receiverCompany);
expressList.setReceiverAddress(receiverAddress);
expressList.setReceiverCode(receiverCode);
int result = JdbcHelper.save(expressList);
if (result > -1) {
    JOptionPane.showMessageDialog(this, "快递单保存成功!");
} else {
    JOptionPane.showMessageDialog(this, "快递单保存失败!");
}
}
```

#### 4.4.6 清空快递单信息

在添加完快递单之后，可以使用“清空”按钮清除快递单中的信息，其事件监听器关键代码如下：

```
clearButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_clearButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_clearButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，其关键代码如下：

```
protected void do_clearButton_actionPerformed(ActionEvent e) {
    senderNameTextField.setText("");
    senderPhoneTextField.setText("");
    senderCompanyTextField.setText("");
    senderAddressTextArea.setText("");
    senderCodeTextField.setText("");
    receiverNameTextField.setText("");
    receiverPhoneTextField.setText("");
    receiverCompanyTextField.setText("");
    receiverAddressTextArea.setText("");
    receiverCodeTextField.setText("");
}
```

这里主要使用文本框和文本域控件的 `setText()` 方法来完成控件内容的清空。

#### 4.4.7 销毁窗体

在使用完该窗体之后，可以单击“返回”按钮销毁该窗体，其事件监听器关键代码如下：

```
returnButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_returnButton_actionPerformed(e);
    }
});
```







在事件监听器中,调用了 do\_returnButton\_actionPerformed()方法,它是由 IDE 工具自动生成的,其关键代码如下:

```
protected void do_returnButton_actionPerformed(ActionEvent e) {
    dispose();
}
```

dispose()方法可以释放该窗体及其子控件占用的资源,该方法的声明如下:

```
public void dispose()
```

## 4.5 修改快递单

### 4.5.1 功能概述

“修改快递单”窗体用于快递信息的浏览和修改。通过单击该窗体上的“上一条”和“下一条”按钮可以浏览快递信息。输入修改后的内容,单击“修改”按钮可以保存修改的快递信息。单击主窗体“快递单管理”/“修改快递单”菜单项,就可以打开“修改快递单”窗体,如图 4.10 所示。



图 4.10 “修改快递单”窗体

### 4.5.2 加载快递单图片

在图 4.10 中显示的快递单是一张图片,通过自定义的 BackgroundPanel 类完成加载,其关键代码如下:

```
// 获得图片资源
URL resource = ExpressUpdateFrame.class.getResource(
    "/image/expressList.jpg");
ImageIcon icon = new ImageIcon(resource); // 利用图片资源创建图标
```





```
// 创建带背景图片的面板
BackgroundPanel backgroundPanel = new BackgroundPanel(icon.getImage());
contentPane.add(backgroundPanel, BorderLayout.CENTER); // 应用面板
```

### 4.5.3 设置文本框和文本域控件

快递单上的信息需要使用文本框和文本域控件进行接收。在加载完快递单图片后，将面板布局管理器设置成 null，然后使用绝对布局来摆放各个控件，其关键代码如下：

```
senderNameTextField = new JTextField(); // 创建发件人姓名文本域
senderNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderNameTextField.setBounds(145, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderNameTextField); // 应用文本域控件
senderNameTextField.setColumns(10); // 设置文本域列数

senderPhoneTextField = new JTextField(); // 创建发件人电话文本域
senderPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderPhoneTextField.setBounds(340, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderPhoneTextField); // 应用文本域控件
senderPhoneTextField.setColumns(10); // 设置文本域列数

senderCompanyTextField = new JTextField(); // 创建发件人公司文本域
// 修改字体
senderCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
senderCompanyTextField.setBounds(145, 146, 325, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderCompanyTextField); // 应用文本域控件
senderCompanyTextField.setColumns(10); // 设置文本域列数

senderAddressTextArea = new JTextArea(); // 创建收件人地址文本域
senderAddressTextArea.setLineWrap(true); // 支持文本域自动换行
senderAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderAddressTextArea.setBounds(145, 190, 325, 80); // 设置文本域控件的位置和大小
backgroundPanel.add(senderAddressTextArea); // 应用文本域控件

senderCodeTextField = new JTextField(); // 创建发件人邮编文本域
senderCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderCodeTextField.setBounds(360, 268, 110, 30); // 设置文本域控件位置和大小
backgroundPanel.add(senderCodeTextField); // 应用文本域控件
senderCodeTextField.setColumns(10); // 设置文本域列数

receiverNameTextField = new JTextField(); // 创建收件人姓名文本域
receiverNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverNameTextField.setBounds(145, 327, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverNameTextField); // 应用文本域控件
receiverNameTextField.setColumns(10); // 设置文本域列数

receiverPhoneTextField = new JTextField(); // 创建收件人电话文本域
// 修改字体
receiverPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
receiverPhoneTextField.setBounds(340, 327, 130, 30); // 设置文本域控件位置和大小
backgroundPanel.add(receiverPhoneTextField); // 应用文本域控件
receiverPhoneTextField.setColumns(10); // 设置文本域列数

receiverCompanyTextField = new JTextField(); // 创建收件人公司文本域
// 修改字体
receiverCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
receiverCompanyTextField.setBounds(145, 356, 325, 30); // 设置文本域控件位置和大小
backgroundPanel.add(receiverCompanyTextField); // 应用文本域控件
receiverCompanyTextField.setColumns(10); // 设置文本域列数

receiverAddressTextArea = new JTextArea(); // 创建收件人地址文本域
receiverAddressTextArea.setLineWrap(true); // 支持文本域自动换行
// 修改字体
```







```
receiverAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15));
// 设置文本区控件的位置和大小
receiverAddressTextArea.setBounds(145, 400, 325, 80);
backgroundPanel.add(receiverAddressTextArea); // 应用文本区控件

receiverCodeTextField = new JTextField(); // 创建收件人邮编文本域
receiverCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverCodeTextField.setBounds(360, 480, 110, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverCodeTextField); // 应用文本域控件
receiverCodeTextField.setColumns(10); // 设置文本域列数
```

#### 4.5.4 添加工具按钮

在窗体中使用了“上一条”、“下一条”、“修改”和“返回”4个按钮，并将它们放置在一个面板中，其关键的代码如下：

```
JPanel buttonPanel = new JPanel(); // 创建按钮面板
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用按钮面板
JButton previousButton = new JButton("上一条"); // 创建“上一条”按钮
// 省略事件处理相关代码
previousButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(previousButton); // 应用按钮
JButton nextButton = new JButton("下一条"); // 创建“下一条”按钮
// 省略事件处理相关代码
nextButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(nextButton); // 应用按钮
JButton modifyButton = new JButton("修改"); // 创建“修改”按钮
// 省略事件处理相关代码
modifyButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(modifyButton); // 应用按钮
JButton returnButton = new JButton("返回"); // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(returnButton); // 应用按钮
```

#### 4.5.5 填充快递单信息

修改是在原来信息基础上进行的修改，因此需要先获取数据库中的快递单信息并将其添加到快递单中，这里使用了 JdbcHelper 类中的 query() 方法获得所有的快递单信息，然后使用自定义的 updateContent() 方法根据索引值添加快递单信息到窗体。该方法的关键代码如下：

```
private void updateContent(int index) {
    ExpressListBean expressList = results.get(index); // 获得第 index 条快递单信息
    senderNameTextField.setText(expressList.getSenderName());
    senderPhoneTextField.setText(expressList.getSenderPhone());
    senderCompanyTextField.setText(expressList.getSenderCompany());
    senderAddressTextArea.setText(expressList.getSenderAddress());
    senderCodeTextField.setText(expressList.getSenderCode());
    receiverNameTextField.setText(expressList.getReceiverName());
    receiverPhoneTextField.setText(expressList.getReceiverPhone());
    receiverCompanyTextField.setText(expressList.getReceiverCompany());
    receiverAddressTextArea.setText(expressList.getReceiverAddress());
    receiverCodeTextField.setText(expressList.getReceiverCode());
}
```





### 4.5.6 获得上一条快递单信息

单击“上一条”按钮，将显示上一条快递单的信息，其事件监听器关键代码如下：

```
previousButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        do_previousButton_actionPerformed(e);  
    }  
});
```

在事件监听器中，调用了 `do_previousButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是第一条记录，则提示用户，其关键代码如下：

```
protected void do_previousButton_actionPerformed(ActionEvent e) {  
    if (index <= 0) {  
        JOptionPane.showMessageDialog(this, "已经是第一条记录!", "",  
JOptionPane.WARNING_MESSAGE);  
        return;  
    }  
    updateContent(--index);           //显示上一条信息  
}
```



说明 index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

### 4.5.7 获得下一条快递单信息

单击“下一条”按钮，将显示下一条快递单的信息，其事件监听器关键代码如下：

```
nextButton.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        do_nextButton_actionPerformed(e);  
    }  
});
```

在事件监听器中，调用了 `do_nextButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的，这里需要对当前位置进行校验。如果已经是最后一条记录，则提示用户，其关键代码如下：

```
protected void do_nextButton_actionPerformed(ActionEvent e) {  
    if (index >= (results.size() - 1)) {  
        JOptionPane.showMessageDialog(this, "已经是最后一条记录!", "",  
JOptionPane.WARNING_MESSAGE);  
        return;  
    }  
    updateContent(++index);           //显示下一条信息  
}
```



说明 index 是指 List 接口中元素的索引，因此是从 0 开始计数。





### 4.5.8 修改快递单信息

通过监听“修改”按钮单击事件，完成对用户修改的快递单信息的保存功能。由于快递单上的信息都非常重要，因此不能为空值。“修改”按钮事件监听器关键代码如下：

```
modifyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_modifyButton_actionPerformed(e);
    }
});
```

do\_modifyButton\_actionPerformed()方法是 IDE 工具生产的方法，它完成了对文本框和文本域的非空校验及保存用户修改信息的功能，其关键代码如下：

```
protected void do_modifyButton_actionPerformed(ActionEvent e) {
    // 获得用户输入的信息并进行非空校验
    String senderName = senderNameTextField.getText();
    if (senderName.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人姓名!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderPhone = senderPhoneTextField.getText();
    if (senderPhone.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人电话!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderCompany = senderCompanyTextField.getText();
    if (senderCompany.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人公司!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderAddress = senderAddressTextArea.getText();
    if (senderAddress.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人地址!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String senderCode = senderCodeTextField.getText();
    if (senderCode.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入发件人邮编!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverName = receiverNameTextField.getText();
    if (receiverName.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人姓名!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverPhone = receiverPhoneTextField.getText();
    if (receiverPhone.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人电话!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverCompany = receiverCompanyTextField.getText();
    if (receiverCompany.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人公司!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String receiverAddress = receiverAddressTextArea.getText();
    if (receiverAddress.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人地址!", "",
            JOptionPane.WARNING_MESSAGE);
    }
}
```





```
        return;
    }
    String receiverCode = receiverCodeTextField.getText();
    if (receiverCode.trim().isEmpty()) {
        JOptionPane.showMessageDialog(this, "请输入收件人邮编!", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    // 将获得的信息保存到 ExpressListBean 中
    ExpressListBean expressList = results.get(index);
    expressList.setSenderName(senderName);
    expressList.setSenderPhone(senderPhone);
    expressList.setSenderCompany(senderCompany);
    expressList.setSenderAddress(senderAddress);
    expressList.setSenderCode(senderCode);
    expressList.setReceiverName(receiverName);
    expressList.setReceiverPhone(receiverPhone);
    expressList.setReceiverCompany(receiverCompany);
    expressList.setReceiverAddress(receiverAddress);
    expressList.setReceiverCode(receiverCode);
    int result = JdbcHelper.update(expressList);
    if (result > -1) {
        JOptionPane.showMessageDialog(this, "快递单修改成功!");
    } else {
        JOptionPane.showMessageDialog(this, "快递单修改失败!");
    }
}
```

## 4.6 打印快递单

### 4.6.1 功能概述

“打印快递单”窗体用于对快递单进行打印及对打印属性进行设置。单击主窗体“打印管理”/“打印快递单”菜单项，就可以打开“打印快递单”窗体，如图 4.11 所示。

图 4.11 “打印快递单”窗体



### 4.6.2 加载快递单图片

在图 4.11 中显示的快递单是一张图片, 通过自定义的 `BackgroundPanel` 类完成加载, 其关键代码如下:

```
// 获得图片资源
URL resource = ExpressUpdateFrame.class.getResource
("/image/expressList.jpg");
ImageIcon icon = new ImageIcon(resource); // 利用图片资源创建图标
// 创建带背景图片的面板
BackgroundPanel backgroundPanel = new BackgroundPanel(icon.getImage());
contentPane.add(backgroundPanel, BorderLayout.CENTER); // 应用面板
```

### 4.6.3 设置文本框和文本域控件

快递单上的信息需要使用文本框和文本域控件进行接收。在加载完快递单图片后, 将面板布局管理器设置成 `null`, 然后使用绝对布局来摆放各个控件, 其关键代码如下:

```
senderNameTextField = new JTextField(); // 创建发件人姓名文本域
senderNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderNameTextField.setBounds(145, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderNameTextField); // 应用文本域控件
senderNameTextField.setColumns(10); // 设置文本域列数

senderPhoneTextField = new JTextField(); // 创建发件人电话文本域
senderPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderPhoneTextField.setBounds(340, 116, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderPhoneTextField); // 应用文本域控件
senderPhoneTextField.setColumns(10); // 设置文本域列数

senderCompanyTextField = new JTextField(); // 创建发件人公司文本域
// 修改字体
senderCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
senderCompanyTextField.setBounds(145, 146, 325, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderCompanyTextField); // 应用文本域控件
senderCompanyTextField.setColumns(10); // 设置文本域列数

senderAddressTextArea = new JTextArea(); // 创建收件人地址文本域
senderAddressTextArea.setLineWrap(true); // 支持文本区自动换行
senderAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderAddressTextArea.setBounds(145, 190, 325, 80); // 设置文本区控件的位置和大小
backgroundPanel.add(senderAddressTextArea); // 应用文本区控件

senderCodeTextField = new JTextField(); // 创建发件人邮编文本域
senderCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
senderCodeTextField.setBounds(360, 268, 110, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(senderCodeTextField); // 应用文本域控件
senderCodeTextField.setColumns(10); // 设置文本域列数

receiverNameTextField = new JTextField(); // 创建收件人姓名文本域
receiverNameTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverNameTextField.setBounds(145, 327, 130, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverNameTextField); // 应用文本域控件
receiverNameTextField.setColumns(10); // 设置文本域列数

receiverPhoneTextField = new JTextField(); // 创建收件人电话文本域
// 修改字体
receiverPhoneTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
receiverPhoneTextField.setBounds(340, 327, 130, 30); // 设置文本域控件的位置和大小
```



```

backgroundPanel.add(receiverPhoneTextField);        // 应用文本域控件
receiverPhoneTextField.setColumns(10);              // 设置文本域列数

receiverCompanyTextField = new JTextField();        // 创建收件人公司文本域
// 修改字体
receiverCompanyTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15));
// 设置文本域控件的位置和大小
receiverCompanyTextField.setBounds(145, 356, 325, 30);
backgroundPanel.add(receiverCompanyTextField);      // 应用文本域控件
receiverCompanyTextField.setColumns(10);            // 设置文本域列数

receiverAddressTextArea = new JTextArea();          // 创建收件人地址文本域
receiverAddressTextArea.setLineWrap(true);         // 支持文本区自动换行
// 修改字体
receiverAddressTextArea.setFont(new Font("微软雅黑", Font.PLAIN, 15));
// 设置文本区控件的位置和大小
receiverAddressTextArea.setBounds(145, 400, 325, 80);
backgroundPanel.add(receiverAddressTextArea);      // 应用文本区控件

receiverCodeTextField = new JTextField();           // 创建收件人邮编文本域
receiverCodeTextField.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 修改字体
receiverCodeTextField.setBounds(360, 480, 110, 30); // 设置文本域控件的位置和大小
backgroundPanel.add(receiverCodeTextField);        // 应用文本域控件
receiverCodeTextField.setColumns(10);              // 设置文本域列数

```

#### 4.6.4 添加工具按钮

在窗体中使用了“上一条”、“下一条”、“打印”和“返回”4个按钮，并将它们放置在一个面板中，其关键的代码如下：

```

JPanel buttonPanel = new JPanel();                // 创建按钮面板
contentPane.add(buttonPanel, BorderLayout.SOUTH); // 应用按钮面板

JButton previousButton = new JButton("上一条");   // 创建“上一条”按钮
// 省略事件处理相关代码
previousButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(previousButton);                  // 应用按钮

JButton nextButton = new JButton("下一条");       // 创建“下一条”按钮
// 省略事件处理相关代码
nextButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(nextButton);                      // 应用按钮

JButton printButton = new JButton("打印");        // 创建“修改”按钮
// 省略事件处理相关代码
printButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(printButton);                     // 应用按钮

JButton returnButton = new JButton("返回");       // 创建“返回”按钮
// 省略事件处理相关代码
returnButton.setFont(new Font("微软雅黑", Font.PLAIN, 15)); // 设置按钮字体
buttonPanel.add(returnButton);                   // 应用按钮

```



#### 4.6.5 填充快递单信息

修改是在原来信息基础上进行的修改，因此需要先获取数据库中的快递单信息并将其添加到快递单中，这里使用了 `JdbcHelper` 类中的 `query()` 方法获得所有的快递单信息，然





后使用自定义的 `updateContent()` 方法根据索引值添加快递单信息到窗体。该方法的关键代码如下：

```
private void updateContent(int index) {
    ExpressListBean expressList = results.get(index); // 获得第 index 条快递单信息
    senderNameTextField.setText(expressList.getSenderName());
    senderPhoneTextField.setText(expressList.getSenderPhone());
    senderCompanyTextField.setText(expressList.getSenderCompany());
    senderAddressTextArea.setText(expressList.getSenderAddress());
    senderCodeTextField.setText(expressList.getSenderCode());
    receiverNameTextField.setText(expressList.getReceiverName());
    receiverPhoneTextField.setText(expressList.getReceiverPhone());
    receiverCompanyTextField.setText(expressList.getReceiverCompany());
    receiverAddressTextArea.setText(expressList.getReceiverAddress());
    receiverCodeTextField.setText(expressList.getReceiverCode());
}
```

#### 4.6.6 获得上一条快递单信息

单击“上一条”按钮，将显示上一条快递单的信息，其事件监听器关键代码如下：

```
previousButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_previousButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_previousButton_actionPerformed()` 方法，它是由 IDE 工具自动生成的。这里需要对当前位置进行校验。如果已经是第一条记录，则提示用户。其关键代码如下：

```
protected void do_previousButton_actionPerformed(ActionEvent e) {
    if (index <= 0) {
        JOptionPane.showMessageDialog(this, "已经是第一条记录！", "",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    updateContent(--index); // 显示上一条信息
}
```



**说明** index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

#### 4.6.7 获得下一条快递单信息

单击“下一条”按钮，将显示下一条快递单的信息，其事件监听器关键代码如下：

```
nextButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_nextButton_actionPerformed(e);
    }
});
```

在事件监听器中，调用了 `do_nextButton_actionPerformed()` 方法，它是由 IDE 工具自





动生成的，这里需要对当前位置进行校验。如果已经是最后一条记录，则提示用户，其关键代码如下：

```
protected void do_nextButton_actionPerformed(ActionEvent e) {
    if (index >= (results.size() - 1)) {
        JOptionPane.showMessageDialog(this, "已经是最后一条记录!", "",
        JOptionPane.WARNING_MESSAGE);
        return;
    }
    updateContent(++index); //显示下一条信息
}
```



说明 index 是指 List 接口中元素的索引，因此是从 0 开始计数的。

#### 4.6.8 分割文本区信息

由于打印时字符串都是绘制到图像中的，并不能直接换行显示，因此需要将较长的信息进行分割处理。这里编写一个 `split()` 方法专门处理文本区中的信息，它以 10 为单位分割给定字符串。分割后的字符串保存在字符串数组中，其关键代码如下：

```
private String[] split(String text) {
    int length = text.length();
    if (length < 10) {
        return new String[] { text };
    } else if (length % 10 == 0) { // 如果字符串长度是 10 的整数倍
        length /= 10; // 截取后字符串的个数是长度除 10 的商
        String[] subStrings = new String[length];
        for (int i = 0; i < subStrings.length; i++) { // 使用循环截取
            subStrings[i] = text.substring(10 * i, 10 * (i + 1));
        }
        return subStrings;
    } else { // 如果字符串长度不是 10 的整数倍
        length = length / 10 + 1; // 截取后字符串的个数是长度除 10 的商再加 1
        String[] subStrings = new String[length];
        // 使用循环截取 10 整数倍的字符串
        for (int i = 0; i < subStrings.length - 1; i++) {
            subStrings[i] = text.substring(10 * i, 10 * (i + 1));
        }
        // 截取尾部剩余的字符串
        subStrings[length - 1] = text.substring(10 * (length - 1), text.length());
        return subStrings;
    }
}
```

#### 4.6.9 打印快递单信息



116



通过监听“打印”按钮，完成打印快递单信息的功能，其事件监听器关键代码如下：

```
printButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        do_printButton_actionPerformed(e);
    }
});
```



do\_printButton\_actionPerformed()方法是 IDE 工具生产的方法,它完成了打印内容的绘制及打印功能,其关键代码如下:

```
protected void do_printButton_actionPerformed(ActionEvent e) {
    PrinterJob job = PrinterJob.getPrinterJob();
    if (job.printDialog()) {
        job.setPrintable(new Printable() {

            @Override
            public int print(Graphics graphics, PageFormat pageFormat, int
                pageIndex) throws PrinterException {
                if (pageIndex > 0) {
                    return Printable.NO_SUCH_PAGE;
                }
                Graphics2D g2d = (Graphics2D) graphics;
                int x = (int) pageFormat.getImageableX();
                int y = (int) pageFormat.getImageableY();
                int height = (int) pageFormat.getImageableHeight();
                int width = (int) pageFormat.getImageableWidth();
                // 绘制背景图片
                g2d.drawImage(icon.getImage(), x, y, width, height, null);
                // // 设置字体
                g2d.setFont(new Font("微软雅黑", Font.PLAIN, 15));
                // 绘制文本信息
                String senderName = senderNameTextField.getText();
                g2d.drawString(senderName, 140, 160);

                String senderPhone = senderPhoneTextField.getText();
                g2d.drawString(senderPhone, 300, 160);

                String senderCompany = senderCompanyTextField.getText();
                g2d.drawString(senderCompany, 140, 190);

                String senderAddress = senderAddressTextArea.getText();
                String[] senderAddresses = split(senderAddress);
                for (int i = 0; i < senderAddresses.length; i++) {
                    g2d.drawString(senderAddresses[i], 140, 230 + i * 30);
                }

                String senderCode = senderCodeTextField.getText();
                g2d.drawString(senderCode, 320, 320);

                String receiverName = receiverNameTextField.getText();
                g2d.drawString(receiverName, 140, 380);

                String receiverPhone = receiverPhoneTextField.getText();
                g2d.drawString(receiverPhone, 300, 380);

                String receiverCompany = receiverCompanyTextField.getText();
                g2d.drawString(receiverCompany, 140, 410);

                String receiverAddress = receiverAddressTextArea.getText();
                String[] receiverAddresses = split(receiverAddress);
                for (int i = 0; i < receiverAddresses.length; i++) {
                    g2d.drawString(receiverAddresses[i], 140, 450 + i * 30);
                }

                String receiverCode = receiverCodeTextField.getText();
                g2d.drawString(receiverCode, 320, 540);

                return Printable.PAGE_EXISTS;
            }
        });
    }
    try {
        job.print(); // 执行打印任务
    } catch (PrinterException e1) {
```



```

        }
        el.printStackTrace();
    }
}

```

打印前的窗体如图 4.12 所示。

图 4.12 打印前的窗体

打印后的效果如图 4.13 所示。

图 4.13 打印后的快递单



打印快递单时使用 A4 纸，同时需要将打印机设置成横向打印。



# 第 5 章

---

## 短信收发模块

( Swing+GSM MODEM 实现 )

在手机已经成为大众交流工具的今天，有关手机的程序开发越来越广泛。本例借助短信猫向单个用户或者多个用户发送手机短信，为商业和企业客户提供方便。

本章对整个短信收发模块做了详细的剖析，将模块划分为几个子模块进行讲解，通过对这几个子模块的学习，读者完全可以掌握短信收发模块的开发技术及原理，笔者通过简洁的程序代码和通俗的技术讲解，使短信收发模块的奥秘完全显示出来，读者可以轻松的理解其中内容。通过本章的学习，读者能够学到：

- » 选项卡的关联
- » 保存配置信息
- » 使用树组件呈现数据
- » 使用表格组件呈现数据
- » 使用文本域组件编辑短信
- » 实现收缩的对话框



## 5.1 短信收发模块概述

### 5.1.1 模块概述

本程序主要利用硬件短信猫发送短信，通过本模块用户可以对指定的一组电话号码进行短信的群发，提高了工作的效率，也可以对回复的项目投票进行统计分析。模块自身还有对已发短信的查询功能及查看 SIM 卡中收到的短信。

为了方便使用，用户可以直接选择联系人中的电话号码，通过这一功能可以轻松添加会员的电话号码，方便在发送短信时添加接收人。

为了减轻用户文字输入工作量，模块提供了常用短语功能，通过此功能用户可以将一些常用的短信短语，在需要时使用，当不需要的时候就可以将其删除。

### 5.1.2 功能结构

短信收发模块包括短信管理、联系人信息管理、常用短语管理和系统等，其功能结构如图 5.1 所示。

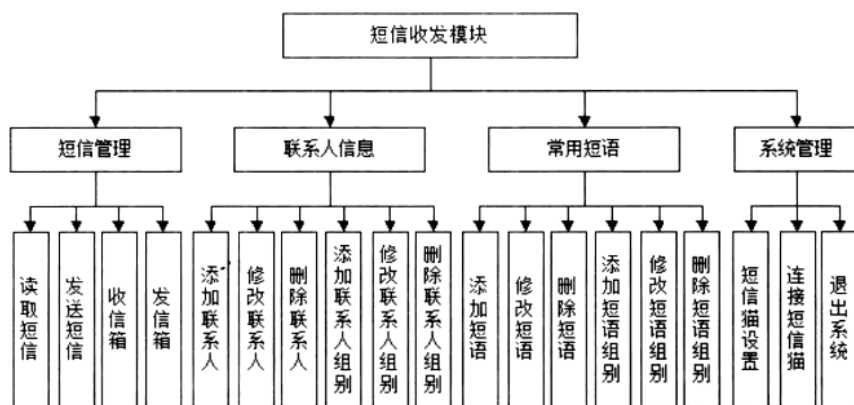


图 5.1 短信收发模块功能结构图

### 5.1.3 程序预览

短信收发模块由多个窗体组成，下面仅列出几个典型窗体，其他窗体参见光盘中的源程序。模块主窗体的运行效果如图 5.2 所示，主要的功能是用于调用执行本模块的所有功能。





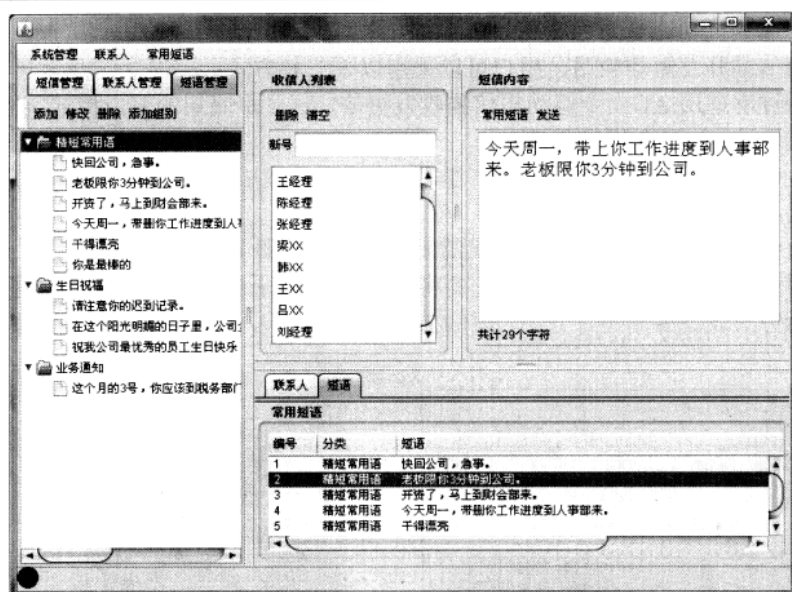


图 5.2 短信收发模块主界面

添加短语对话框的效果如图 5.3 所示，其主要功能是向数据库添加常用短语信息，并将其分类存放，在发送短信时可以方便调用。

添加联系人对话框的运行效果如图 5.4 所示，主要功能用于添加联系人信息，并将其分类管理。该界面分为普通信息和详细信息，其中详细信息是可选的。

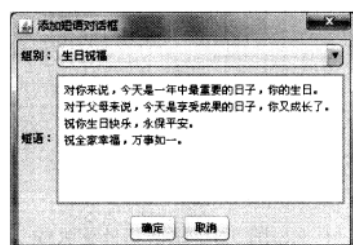


图 5.3 添加短语界面

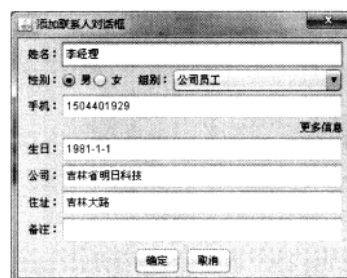


图 5.4 添加联系人界面

## 5.2 关键技术

### 5.2.1 短信猫技术

本模块的核心依赖于短信猫技术来实现短信的接收和发送，所谓短信猫，其实是一种工业级 GSM MODEM，通过串口与计算机连接，可以通过 AT 指令控制进行短信收发的设备。国内目前应用较多的短信猫，都是以 SIEMENS 或 WAVECOM 模块为核心组装而



成的，与普通手机相比更为稳定高效。

基于短信猫开发短信应用，用户可以采用以下三种方式：

- 通过串口用 AT 指令驱动短信猫收发短信，这是最底层的开发模式，需要对短信模块的 AT 指令相当熟悉。
- 短信猫开发包：短信猫厂商基于串口 AT 指令集成的应用开发包，用户只需直接调用短信收发 API 即可。
- 短信中间件：短信猫厂商提供的短信收发后台服务软件。

在使用短信猫收发短信之前，首先要将短信猫设备与计算机进行连接，下面以金仓短信王标准商务版设备为例进行介绍。短信猫如图 5.5 所示。专用电源充电器如图 5.6 所示。

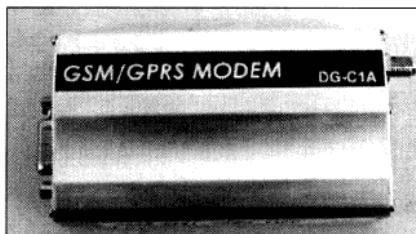


图 5.5 短信猫

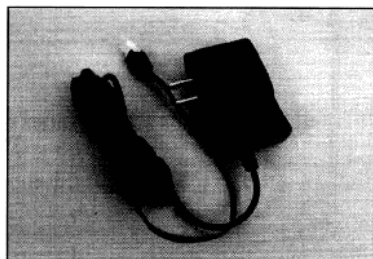


图 5.6 短信猫充电器

通信线：串口数据通信线如图 5.7 所示。  
短信猫天外置的天线如图 5.8 所示。



图 5.7 短信猫通信线

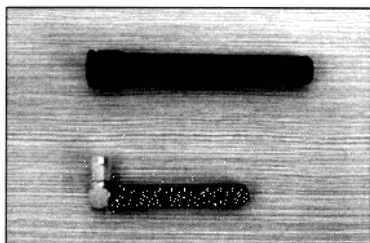


图 5.8 短信猫外置天线

安装 SIM 卡：用圆珠笔或其他工具顶 SIM 卡座按钮，设备会弹出 SIM 卡座，将 SIM 卡放入卡座，如图 5.9 所示，再将卡座插回设备的 SIM 卡插孔即可，如图 5.9~图 5.11 所示。安装天线：在设备后面有一铜螺柱，将天线拧上，如图 5.12 所示。

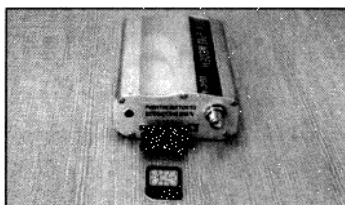


图 5.9 短信猫的 SIM 卡座

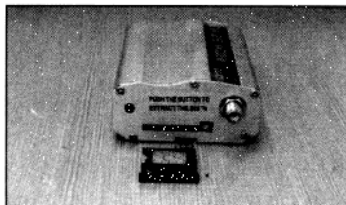


图 5.10 将 SIM 卡放入卡座





图 5.11 将 SIM 卡座插回短信猫

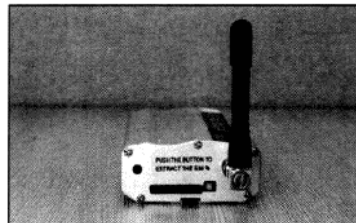


图 5.12 安装短信猫天线

连接串口线:将圆形插针与 DG-C1A 相连,另一头为 9 针 DB 插座与计算机相连即可,如图 5.13 和图 5.14 所示。

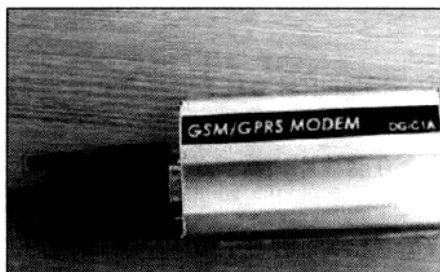


图 5.13 数据线与短信猫连接

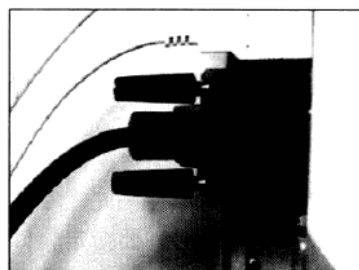


图 5.14 数据线与计算机串口相连接

连接电源:将电源线的圆形插孔插入设备的电源插孔,最后将电源插座插到电源插线板上即可,如图 5.15 所示。

通过指示灯检查连接状态:正确连接指示灯为红灯间歇闪烁,如图 5.16 所示。



图 5.15 将电源线插到短信猫电源插孔



图 5.16 通过指示灯检查连接状态

当短信猫安装完毕之后,接下来便是通过程序操作短信猫,在购买短信猫时会附带有 SDK 的开发包,其中提供了操作短信猫的函数(封装在 BestMail.jar 库中)。

## 5.2.2 收发短信

在本模块中使用短信猫设备实现短信的收发业务,短信猫需要经过设置、连接等初始化工作,然后才能调用发送和接收短信的方法。本节将介绍从短信猫初始化及连接到收发短信的工作流程。





### 1. 设置短信猫

在利用短信猫发送短信时，首先需要对短信猫进行设置：包括端口号、波特率、通信许可证的序列号等注册信息和连接硬件，利用厂商提供的 API，获取短信发送和接收所需要的相关信息。

这些 API 都封装在 BestMail.jar 库文件中，该文件可以由厂家短信猫驱动光盘中获得，在设置短信猫时涉及如下操作。

#### 1) 获取短信猫注册需要的信息

获取短信猫的注册信息可以通过 GSMModemGetSnInfoNew()方法实现，获取的注册标识码可以从厂商获取授权码。语法格式如下：

```
public native String GSMModemGetSnInfoNew(String device,      //端口号
                                           String baudrate);   //波特率
```



说明 device 是 String 类型，表示通信端口，为 null 时系统会自动检测。baudrate

是 String 类型，表示通信波特率，为 null 时系统会自动检测。返回值是 String 类型，表示短信标识码，将此号码发送给厂商即可获得正式的授权码。

在调用该方法时需要指定端口号和波特率参数，关键代码如下：

```
private void linkMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    smssend = new smssend();
    // 获取注册码信息
    String reginfo = smssend.GSMModemGetSnInfoNew(device, baud);
    if (reginfo == null) {
        System.out.println("GSMModemGetSnInfo: connect failed! Error="
+smssend.GSMModemGetErrorMsg());
        JOptionPane.showMessageDialog(null, "短信猫注册失败");
        return;
    }
    System.out.println("GSMModemGetSnInfo=" + reginfo);
    .....// 省略其他代码
}
```

#### 2) 初始化 GSM MODEM 设备

在操作短信猫收发短信之前，必须调用 GSMModemInitNew()方法初始化硬件设备，语法格式如下：

```
public native boolean GSMModemInitNew(
    String device,      // 端口号
    String baudrate,    // 波特率
    String initstring,  // at 初始化命令
    String charset,     // 与 GSM MODEM 通信的字符集 GSM、UCS2
    boolean swHandshake, // 软件握手
    String sn);         // 通信许可证书
```

该方法的参数说明如表 5.1 所示。

表 5.1 GSMModemInitNew 方法的参数说明

参数	类型	描述
device	String	通信口，为 null 时系统会自动检测
baudrate	String	通信波特率，为 null 时系统会自动检测
initstring	String	通信字符集，设为 null，系统默认即可







续表

参数	类型	描述
charset	String	通信字符集, 设为 null, 系统默认即可
swHandshake	Boolean	是否进行软件握手, 设为 false 即可
Sn	String	通信许可证书, 区分大小写, 例如 "XXXX-XXXX-XXXX-XXXX"
返回值	Boolean	true 为成功, false 连接失败

使用该方法初始化短信猫设备的关键代码如下:

```
private void linkMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    smssend = new smssend();
    .....// 省略部分代码
    // 初始化短信猫设备
    if (!smssend.GSMModemInitNew(device, baud, null, null, false, sn)) {
        System.out.println("GSMModemInit: connect failed! Error=" +
            smssend.GSMModemGetErrorMsg());
        JOptionPane.showMessageDialog(null, "短信猫连接失败");
        linkLabel1.setColor(Color.RED);
    } else {
        System.out.println("初始化成功");
        linkLabel1.setColor(Color.GREEN);
    }
}
```

## 2. 利用短信猫发送短信

在设置好短信猫以后, 就可以发送短信了。在短信发送时, 它能够对输入或选择的电话号码发送短信, 系统首先连接硬件“短信猫”, 然后在模块中通过程序控制短信的发送。

在短信收发模块中主要用到的技术是厂商提供的 BestMail.jar 库中的 GSMModemSMSsend()方法, 该方法用于将输入的短信通过短信猫发送到指定的接收号码中, 其中该方法的参数及说明如下:

```
public native boolean GSMModemSMSsend(
    String serviceCenterAddress, //短信中心号码
    int codeval,                 //文本编码格式: 0-7bit;
                                //4-8bit, 8-16bit
    byte [] text,                //发送文本
    String phonenumber,          //电话号码
    boolean requestStatusReport); //状态报告
```

其参数说明如表 5.2 所示。

表 5.2 GSMModemSMSsend 方法的参数说明

参数		说明
serviceCenterAddress	String	字符串型值, 短信中心号码
encodeval	Int	文本编码格式: 0-7bit, 4-8bit, 8-16bit (1) 中文短信时, 使用编码 8, 其短信长度支持 70 个汉字 (2) 纯英文短信时, 可以使用编码 4, 其短信长度支持 160 个字节
text	byte[]	表示短信内容
phonenumber	String	接收电话号码
requestStatusReport	boolean	状态报告



125





使用短信猫 API 发送短信的关键代码如下：

```
byte[] data = smssend.getUNIByteArray(mes);
smssend.GSMModemSMSsend(null, 8, data, phoneNum, false);
System.out.println("电话: " + phoneNum + "\t内容: " + mes);
```

### 3. 短信猫接收短信

使用短信猫除了可以发送短信外，还可以接收短信。在接收短信时，首先要对硬件短信猫进行连接，连接成功之后判断短信猫中的 SIM 卡中是否有短信，如果有短信，则从 SIM 卡中将短信读取出来保存在数据库中，然后通过数据控件将短信内容显示出来。

在短信接收模块中主要用到的技术是厂商提供的 BestMail.jar 库中的 GSMModemSMSReadAll()方法，该方法用于接收短信猫中 SIM 卡里的短信内容，方法的语法形式如下：

```
public native String[] GSMModemSMSReadAll(int selectOper);
```

其中，selectOper 参数为整型，表示对读取的短信息处理，0 表示删除，1 表示不处理，其返回值为短消息，是一个字符串型数组，返回的短信内容（即数组的每个元素）格式为：

电话号码 | 短信内容

使用短信猫读取短信数据的关键代码如下：

```
public void readSMS() {
    String[] infos = smssend.GSMModemSMSReadAll(1);           // 读取短信内容
    if (infos == null) {
        return;
    }
    .....// 解析短信字符串
}
```

## 5.2.3 选项卡的关联

主窗体中包含两个选项卡组件，它们分别位于主窗体左侧和写短信面板的下方。主窗体左侧的选项卡包括“短信管理”、“联系人管理”和“短语管理”，主要用于管理面板的切换。而写短信界面下方的选项卡包括“联系人”和“短语”，主要用于切换选择联系人和选择常用短语的切换。这两个选项卡如图 5.17 所示。

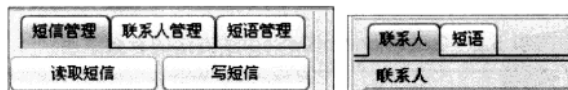


图 5.17 主窗体左侧的选项卡、写短信界面下方的选项卡效果图



126



这两个选项卡中左侧选项卡的“联系人管理”、“短语管理”应该是和右侧选项卡“联系人”、“短语”绑定的，即单击其中一个选项卡的任何一个选项，例如“联系人管理”，另一个选项卡应该自动选择相应的选项，例如“联系人”，反之亦然。这样在左侧的“联系人管理”或“短语管理”选择相应的联系人分类或短语分类，在写短信界面会显示相应组别的联系人和短语信息。这样会为用户发送短信提供方便。

实现原理：当用户单击主窗体左侧的管理选项卡时，判断选择的是哪一个选项，并控制另一个界面的选项卡，选择相应的选项，关键代码如下：



```

private void manageTabbedPaneStateChanged(javax.swing.event.ChangeEvent evt)
{
    int index = manageTabbedPane.getSelectedIndex(); // 获取选项卡选择的索引号
    CardLayout layout;
    switch (index) {
        case 0:
            System.out.println("select 短信管理");
            break;
        case 1: // 如果索引是1, 即“联系人管理”选项卡
            // 切换发送面板的联系人选项卡
            System.out.println("联系人管理");
            // 调用 setSelectedLxrTabbedPane() 方法, 切换选项卡
            sendPanel.setSelectedLxrTabbedPane();
            initLxrTree(); // 初始化联系人管理的树组件
            break;
        case 2:
            System.out.println("短语管理"); // 如果选择了“短语管理”选项卡
            // 调用 setSelectedNoteTabbedPane() 方法切换选项卡
            sendPanel.setSelectedNoteTabbedPane();
            initDuanyuTree(); // 初始化短语管理的树组件
            break;
        default:
            break;
    }
}

```

## 5.2.4 卡片布局

主窗体右侧的界面使用了 Java Swing 的 CardLayout 卡片布局, 并叠加布局 “SendNotePanel” 发送短信面板和 “NotePanel” 短信面板。CardLayout 布局管理器, 使它们像卡片一样叠放在一起, 而且为每个卡片起了不同的名字, 当需要使用哪个面板时就将其置前并显示在所有卡片的最上层。使用卡片布局可以灵活运用界面空间, 实现最理想的布局。卡片布局就像一个名片盒一样, 将所有组件当做一个卡片处理, 同一时间只能显示一张卡片 (或名片), 如图 5.18 所示。

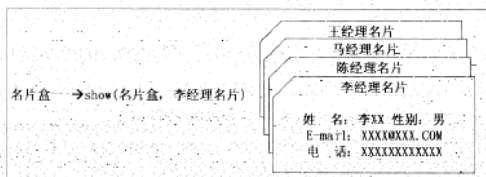


图 5.18 卡片布局管理器示意图

卡片布局管理器调用 show 方法指定显示组件的名称, 并使该组件显示在所有卡片的最顶层。

### 1. 卡片布局管理器的构造方法

CardLayout 类, 也就是卡片布局管理器包含两个构造方法:

#### 1) CardLayout()

创建一个间距大小为 0 的新卡片布局, 其语法形式如下:

```
CardLayout layout=new CardLayout();
```



### 2) CardLayout(int hgap, int vgap)

创建一个具有指定水平间距和垂直间距的新卡片布局。水平间距置于左右边缘，垂直间距置于上下边缘。

其语法形式如下：

```
CardLayout layout=new CardLayout(int hgap, int vgap);
```

□ hgap: 水平间距。

□ vgap: 垂直间距。

### 2. CardLayout 类的常用方法

#### 1) first()方法

该方法用于翻转到容器的第一张卡片，其语法形式如下：

```
public void first(Container parent)
```

parent: 必要的参数，由卡片布局管理器管理的父容器。

#### 2) previous()方法

翻转到指定容器的前一张卡片。如果当前的可见卡片是第一个，则此方法翻转到布局的最后一张卡片，其语法形式如下：

```
public void previous (Container parent)
```

parent: 必要的参数，由卡片布局管理器管理的父容器。

#### 3) last()方法

该方法用于翻转到容器的最后一张卡片，其语法形式如下：

```
public void last (Container parent)
```

parent: 必要的参数，由卡片布局管理器管理的父容器。

#### 4) addLayoutComponent()方法

将指定的组件添加到此卡片布局的内部名称表。通过调用 show 方法，应用程序可以显示具有指定名称的组件，其语法形式如下：

```
public void addLayoutComponent(Component comp, Object constraints)
```

□ comp: 要添加的组件。

□ constraints: 标识布局中特定卡片的标记。



constraints 指定的对象必须是一个字符串。卡片布局将此字符串作为一个键

一值对存储起来，该键一值对可用于对特定卡片的随机访问。

#### 5) show()方法

翻转到使用 addLayoutComponent ()方法添加到此布局的具有指定名称的组件，其语法形式如下：

```
public void show(Container parent, String name)
```

□ parent: 必要的参数，由卡片布局管理器管理的父容器。

□ name: 组件在布局管理器中的名称。

例如，本模块中的“写短信”按钮的事件处理方法中，就调用了 Show()方法显示发送短信面板，关键代码如下。

```
private void writeNoteButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // 获取卡片布局管理器
```



128







```
CardLayout layout = (CardLayout) contentPanel.getLayout();
layout.show(contentPanel, "发送短信面板"); // 调用 show 方法显示指定的面板
}
```

### 5.2.5 树控件的使用

在主窗体的左侧的选项卡控件中,“联系人”和“短信”选项面板中各自包含了一个树组件,Java 的数组件由 JTree 类创建,它们可以灵活地显示联系人和短信信息,最重要的是使用数组件可以方便地显示各种信息的分类及子分类,选项卡的每个节点和子节点都是 DefaultMutableTreeNode 类的实例对象。该对象可以设置是否允许有子节点,从而判断选择的树节点是联系人和短信信息还是相关的分类(组别)。显示联系人信息的树组件界面如图 5.19 所示。



图 5.19 联系人管理界面的树组件

本模块通过自定义的 initLxrTree()方法,从数据库中获取联系人信息,初始化了联系人管理面板的树组件,关键代码如下:

```
public void initLxrTree() {
    DefaultTreeModel model = (DefaultTreeModel) lxrTree.getModel();
    DefaultMutableTreeNode root = (DefaultMutableTreeNode) model.getRoot();
    root.removeAllChildren(); // 清除树组件的原始内容
    Dao dao = Dao.getDao(); // 初始化数据库操作类
    List lxrSuite = dao.getLxrSuite(); // 获取数据库中的联系人记录
    if (lxrSuite == null) {
        return;
    }
    Iterator iterator = lxrSuite.iterator();
    while (iterator.hasNext()) { // 迭代遍历联系人记录
        String suite = (String) iterator.next(); // 获取组别信息
        // 使用组别名称创建树的节点,并允许有子节点
        DefaultMutableTreeNode node = new DefaultMutableTreeNode(suite, true);
        List lxrBeans = dao.getLxrBeans(suite); // 获取该组别的联系人信息
        Iterator lxrIterator = lxrBeans.iterator();
        while (lxrIterator.hasNext()) { // 使用联系人信息创建子节点
            TbLinkman lxr = (TbLinkman) lxrIterator.next();
            node.add(new DefaultMutableTreeNode(lxr, false));
        }
        root.add(node);
    }
    model.reload(); // 重新装载树模型
    lxrTree.setSelectionRow(0);
}
```

## 5.3 设置并连接短信猫



### 5.3.1 功能概述

程序启动之后的第一件事,就是选择“系统管理”/“短信猫设置”菜单项,然后在弹出的如图 5.20 所示的对话框中设置短信猫的参数,只有提供了正确的参数,并选择“系



统管理”/“连接”菜单项进行短信猫的连接，才能正确地实现短信收发业务。

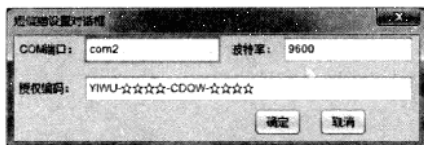


图 5.20 短信猫设置对话框



图中授权编码的“☆”字符不是合法的授权编码，是本书保护合法授权编码的一种措施而替换的字符。

## 5.3.2 短信猫设置

短信猫设置对话框中的“COM 端口”、“波特率”、“授权编码”等文本框组件的内容都是从数据库中读取，然后显示在界面中的。要实现窗口创建的同时就初始化这些组件的文本内容，需要编写实现该对话框的 NoteSetDialog 类的构造方法，在构造方法中读取数据库内容并赋值到组件中。NoteSetDialog 类构造方法的关键代码如下：

```
public NoteSetDialog(NoteFrame parent, boolean modal) {  
    super(parent, modal);           // 执行超类的构造方法  
    this.frame = parent;  
    initComponents();               // 初始化程序界面的方法  
    setTitle("短信猫设置对话框");  
    String[] sets = Dao.getDao().getSetting(); // 从数据库获取设置信息  
    if (sets != null) {             // 如果数据库存在设置信息  
        snTextField.setText(sets[0]); // 设置界面组件的文本内容  
        comTextField.setText(sets[1]);  
        baudTextField.setText(sets[2]);  
    }  
}
```

在这个构造方法中调用了 Dao 类的 getSetting()方法获取数据库中的短信猫设置信息，该方法将所有设置信息以字符串数组的形式返回给方法的调用者，程序关键代码如下：

```
public String[] getSetting() {  
    String[] sets = null;           // 定义字符串数组  
    try {  
        // 定义查询 tb_setting 表数据的 SQL 语句  
        String sql = "select sn,com,baud from tb_setting where id=1";  
        // 执行 SQL 语句获取 tb_setting 表的查询结果集  
        ResultSet rs = conn.createStatement().executeQuery(sql);  
        if (rs.next()) {             // 如果存在数据  
            // 初始化字符串数组  
            sets = new String[] { rs.getString("sn"), rs.getString("com"),  
                                   rs.getString("baud") };  
        }  
    } catch (SQLException ex) {  
        Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);  
    }  
    return sets;                     // 返回查询结果  
}
```







当添加或修改了对话框中对短信猫的设置之后,单击“确定”按钮,将修改短信猫的设置,该业务由 `okButtonActionPerformed()` 方法完成,该方法首先获取界面中的设置信息,然后把设置信息同步到主窗体,最后把这些设置通过 `Dao` 类的 `updateSetting()` 方法保存到数据库中。关键代码如下:

```
// 确定按钮的业务方法
private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String sn = snTextField.getText();           // 获取授权编码
    String com = comTextField.getText();         // 获取COM端口号
    String baud = baudTextField.getText();       // 获取波特率
    frame.sn = sn;                              // 同步主类的短信猫设置
    frame.device = com;
    frame.baud = baud;
    Dao.getDao().updateSetting(sn, com, baud);   // 更新数据库中的设置信息
    dispose();                                  // 销毁对话框窗体
}
```

`Dao` 类的 `updateSetting()` 方法负责保存短信猫的设置信息到数据库中,该方法首先会判断 `tb_setting` 数据表中是否有短信猫设置的记录,如果没有,就执行插入语句,添加新的数据,否则生成更新语句,修改原有的配置信息。程序代码如下:

```
public void updateSetting(String sn, String com, String baud) {
    String sql = "select * from tb_setting where id=1";
    try {
        // 查询数据库中的设置信息
        ResultSet rs = conn.createStatement().executeQuery(sql);
        if (rs.next()) {                      // 如果有设置信息
            // 生成 update 更新语句
            sql = "update tb_setting set sn=?,com=?,baud=? where id=1";
        } else {                             // 否则
            sql = "insert into tb_setting values(1,?,?,?)"; // 生成 insert 插入语句
        }
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, sn);                 // 设置 SQL 语句的参数
        ps.setString(2, com);
        ps.setString(3, baud);
        ps.execute();                         // 执行 SQL 语句
    } catch (SQLException ex) {
        Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

### 5.3.3 连接短信猫

在主菜单中选择“系统管理”/“连接”菜单项,将执行短信猫的连接。在窗体最下方的状态栏中有一个圆形图标,当它呈现为绿色圆形时,说明短信猫连接成功;如果呈现为红色,说明,短信猫连接失败。

连接短信猫的业务由 `linkMenuItemActionPerformed()` 方法实现,该方法将有“连接”菜单项的事件处理器调用,当单击该菜单项时,将触发这个事件处理器并调用该方法完成短信猫的连接。初始化并连接短信猫的关键代码如下:

```
private void linkMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    smssend = new smssend();
    // 获取注册码信息
    String reginfo = smssend.GSMModemGetSnInfoNew(device, baud);
    if (reginfo == null) {
        System.out.println("GSMModemGetSnInfo: connect failed! Error="
```





```
        + smssend.GSMModemGetErrorMsg());
JOptionPane.showMessageDialog(null, "短信猫注册失败");
return;
}
System.out.println("GSMModemGetSnInfo=" + reginfo);
// 初始化短信猫设备
if (!smssend.GSMModemInitNew(device, baud, null, null, false, sn)) {
    System.out.println("GSMModemInit: connect failed! Error="
        + smssend.GSMModemGetErrorMsg());
    JOptionPane.showMessageDialog(null, "短信猫连接失败");
    linkLabel1.setColor(Color.RED); // 设置连接状态为红色
} else {
    System.out.println("初始化成功");
    linkLabel1.setColor(Color.GREEN); // 设置连接状态为绿色
}
}
```

## 5.4 读取短信

### 5.4.1 功能概述

在主窗体左侧的选项卡组件主要实现管理功能，包括“短信管理”、“联系人管理”和“短语管理”选项卡，每个选项卡由不同的组件面板呈现，其中“短信管理”选项卡包括“读取短信”、“写短信”和“发短信功能”。

在“短信管理”选项卡中单击“读取短信”按钮，将执行短信读取业务，程序会从短信猫中读取短信，并根据短信的数据格式进行解析、编码，然后保存到数据库的 tb\_sms 数据表中。单击“读取短信”按钮之后的程序界面如图 5.21 所示。

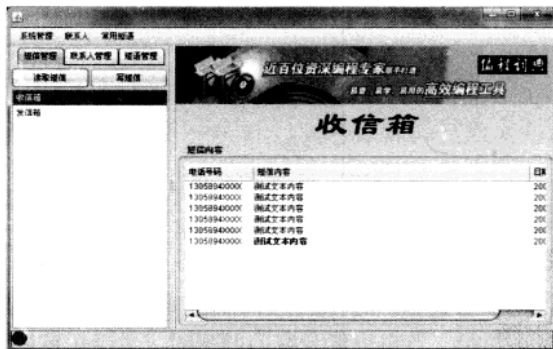


图 5.21 读取短信的界面

### 5.4.2 读取短信

“读取短信”按钮的业务由 readButtonActionPerformed() 方法实现，该方法将调用 readSMS() 方法从短信猫中读取短信，读取的短信内容会保存到数据库中，然后使短信管理列表组件默认选择第一项，并触发相应的事件方法完成显示短信内容的业务。读取短信





的关键代码如下:

```
// 处理读取短信按钮业务的方法
private void readButtonActionPerformed(java.awt.event.ActionEvent evt) {
    noteManageList.setSelectedIndex(0);
    readSMS(); // 调用读取短信的方法
    // 调用处理列表选择事件的方法
    noteManageListValueChanged(new
    ListSelectionEvent(noteManageList,0,0,false));
}
```

该按钮的业务处理代码中调用了 readSMS()方法,这个方法主要负责读取短信文本和电话号码,并保存到数据库中,程序关键代码如下:

```
public void readSMS() {
    String[] infos = smssend.GSMModemSMSReadAll(1); // 读取短信内容
    if (infos == null) {
        return;
    }
    for (int i = 0; infos != null && i < infos.length; i++) {
        String[] sms = infos[i].split("#"); // 解析短信字符串
        String info = "";
        if (sms[1].equalsIgnoreCase("8")) { // 如果类型是文字短信
            info = smssend.HexToBuf(sms[2]); // 解码短信文本
        } else { // 否则
            try { // 直接获取短信内容
                info = new String(sms[2].getBytes(), "gb2312");
            } catch (UnsupportedEncodingException ex) {
                Logger.getLogger(NoteFrame.class.getName()).log(
                    Level.SEVERE, null, ex);
            }
        }
        SmsBean smsbean = new SmsBean(); // 创建短信实体对象
        smsbean.setPhoneNum(sms[0]); // 初始化短信实体
        smsbean.setSmsText(info);
        smsbean.setSmsDate(new Date(System.currentTimeMillis()));
        Dao.getDao().addSms(smsbean); // 保存短信实体到数据库
        System.out.println(sms[0] + "-----" + sms[1] + info);
    }
}
```

### 5.4.3 显示短信

读取短信的内容不但需要保存到数据库,同时还要将这些信息显示到程序界面中,所以“读取”按钮的事件处理方法中,触发了短信管理列表的选择事件,也就是说,创建一个 ListSelectionEvent 选择事件,这个事件对象的参数指定选择列表的第一个,也就是“收信箱”选项;然后由列表组件处理这个事件,把短信内容显示到窗体中。处理这个事件的程序关键代码如下:

```
private void noteManageListValueChanged(
    javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        int index = noteManageList.getSelectedIndex();
        if (index == 0) {
            notePanel.setTitle("收信箱"); // 如果选择完毕
            notePanel.setLxrPanelShow(false); // 获取选择索引
            List smss = Dao.getDao().getSmss(); // 如果选择收信箱选项
            notePanel.setReceiveSmsInfo(smss); // 设置短信面板标题
        } else { // 不显示联系人列表
            notePanel.setTitle("发信箱"); // 获取短信数据集
        } // 显示短信数据到表格中
        // 如果选择发信箱选项
    }
}
```





```

        notePanel.setLxrPanelShow(true);           // 显示联系人列表
        List infos = Dao.getDao().getSmsSendInfo(); // 获取发送短信的数据
        notePanel.setSendSmsInfo(infos);           // 显示这些数据
    }
    notePanel.revalidate();
    CardLayout layout = (CardLayout) contentPanel.getLayout();
    layout.show(contentPanel, "短信收发记录");     // 显示收发短信面板或卡片
}

```

## 5.5 发送短信

### 5.5.1 功能概述

在“短信管理”选项卡中，单击“写短信”按钮，卡片布局管理器将在窗体右侧显示编写短信的面板组件，或者说是卡片。如图 5.22 所示，在这个界面中可以通过“联系人管理”选项卡和“短语管理”选项卡添加联系人到“收信人列表”区域，或添加短语到“短信内容”区域中。最后，在“短信内容”区域中单击“发送”按钮，将执行短信发送命令。

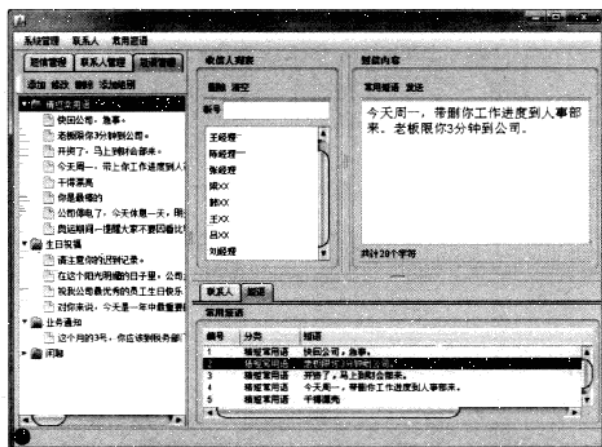


图 5.22 发送短信的界面

### 5.5.2 添加删除收信人

短信发送功能的“收信人列表”区域包括“删除”、“清空”按钮；添加新号的文本框和显示联系人的列表组件。下面分别介绍这些组件的实现。

(1) 单击“删除”按钮将删除联系人列表中当前选择的选项，关键代码如下：

```

private void delButtonActionPerformed(java.awt.event.ActionEvent evt) {
    int[] sels = phoneList.getSelectedIndices(); // 获取列表选择的多个索引
    DefaultListModel model = (DefaultListModel) phoneList.getModel();
    for (int i = sels.length - 1; i >= 0; i--) { // 遍历索引数组
        model.remove(i); // 倒序溢出选择内容
    }
}

```



```

    }
}

```

(2) “清空”的主要用途也是删除联系人列表(或收信人列表)的选项,但是与“删除”按钮不同的是,它将删除所有联系人信息,关键代码如下:

```

private void clearButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 获取列表组件的数据模型
    DefaultListModel model = (DefaultListModel) phoneList.getModel();
    model.removeAllElements(); // 移除模型的所有元素
}

```

(3) 添加新号码的文本框用于在联系人列表之外添加新的联系电话号码,也就是临时增加数据库中没有的联系人电话号码。在该文本框中只能输入数字,通过按键盘的【Enter】键执行添加操作。程序关键代码如下:

```

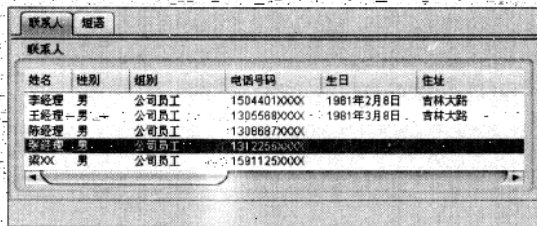
private void newPhoneTextFieldKeyPressed(java.awt.event.KeyEvent evt) {
    char key = evt.getKeyChar(); // 获取输入字符
    if (key == '\n') { // 如果是回车字符
        String phone = newPhoneTextField.getText(); // 获取文本框中的电话号码
        if (phone == null || phone.isEmpty()) {
            return;
        }
        // 获取列表组件模型
        DefaultListModel model = (DefaultListModel) phoneList.getModel();
        model.addElement(phone); // 为模型添加新的电话号码
        newPhoneTextField.setText(""); // 清除文本框内容
    }
}

private void newPhoneTextFieldKeyTyped(java.awt.event.KeyEvent evt) {
    String keys = "0123456789" + (char) 8; // 定义约束调节字符串
    if (keys.indexOf(evt.getKeyChar()) == -1) { // 如果输入字符不符合约束条件
        evt.consume(); // 销毁本次输入的字符
    }
}

```

### 5.5.3 “联系人”选项卡

“联系人”选项卡由一个表格组件填充界面,这个表格的内容根据主窗体左侧的“联系人管理”选项卡中不同的选择而改变。例如,选择不同的组别或具体联系人。但是要向联系人列表(收信人列表)添加联系人,就必须通过双击“联系人”选项卡的表格中选择的内容进行添加,如图 5.23 所示。



姓名	性别	组别	电话号码	生日	住址
李松理	男	公司员工	15044010000	1981年2月8日	吉林大街
王松理	男	公司员工	13055800000	1981年3月8日	吉林大街
陈松理	男	公司员工	13066870000		
张松理	男	公司员工	13122580000		
周XX	男	公司员工	15911250000		

图 5.23 发送短信的联系人选项卡界面

在联系人表格中选择一个联系人,通过双击鼠标可以将该联系人添加到“收信人列表”区域的列表组件中,程序关键代码如下:





```

private void lxrTableMouseClicked(java.awt.event.MouseEvent evt) {
    if (evt.getClickCount() % 2 == 0) { // 判断鼠标是否双击
        int row = lxrTable.getSelectedRow(); // 获取表格当前选择行
        if (row < 0) {
            return;
        }
        // 获取表格当前行的第一列数据，它实际上是联系人实体对象
        TbLinkman lxr = (TbLinkman) lxrTable.getValueAt(row, 0);
        // 获取联系人列表的数据模型
        DefaultListModel model = (DefaultListModel) phoneList.getModel();
        model.addElement(lxr); // 添加联系人到列表中
    }
}

```

### 5.5.4 编写短信内容

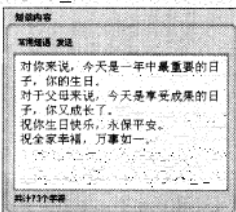


图 5.24 发送短信的短信内容区域

在发送短信界面的“短信内容”区域用于短信内容的编写，它主要包括“常用短语”和“发送”按钮，还有一个最重要的编写短信内容的文本域，如图 5.24 所示。

在界面中单击“常用短语”按钮，将切换下面的选项卡组件为“短语”选项卡界面，简单地调用 `setSelectNoteTabbed()` 方法即可实现，关键代码如下：

```

private class NoteButtonActionListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        setSelectNoteTabbed(); // 切换“短语”选项卡
    }
}

```

`setSelectNoteTabbed()` 方法用于切换选项卡到“短语”界面，程序的关键代码如下：

```

void setSelectNoteTabbed() {
    infoTabbed.setSelectedComponent(noteTabPanel);
}

```

在文本域组件中输入，或从常用短语中添加短信内容，然后单击“发送”按钮，将执行短信发送命令，这是由 `sendButtonActionPerformed()` 事件处理方法完成的，关键代码如下：

```

private void sendButtonActionPerformed(java.awt.event.ActionEvent evt) {
    NoteFrame frame = (NoteFrame) getRootPane().getParent();
    String text = noteTextArea.getText(); // 获取短信内容
    if (text == null || text.isEmpty()) {
        return;
    }
    DefaultListModel model = (DefaultListModel) phoneList.getModel();
    Object[] toArray = model.toArray(); // 获取收信人列表
    if (toArray.length < 1) {
        return;
    }
    TbSendsms sendsms = new TbSendsms(); // 创建短信发送表的实体对象
    List sublist = new ArrayList(); // 创建收信人集合
    sendsms.setTbSendsubCollection(sublist); // 将该集合添加到短信实体
    sendsms.setSmsdate(new Date(System.currentTimeMillis())); // 初始化短信实体对象
    sendsms.setSmsText(text); // 初始化短信内容
    for (int i = 0; i < toArray.length; i++) { // 遍历收信人列表
        TbSendsub sub = new TbSendsub(); // 创建收信人数据表的实体对象
    }
}

```







```
String phone;
if (toArray[i] instanceof TbLinkman) { // 如果列表元素是联系人实体对象
    phone = ((TbLinkman) toArray[i]).getPhonenum(); // 获取电话号码
} else { // 否则
    phone = toArray[i].toString(); // 调用元素的 toString() 方法
}
frame.sendSMS(phone, text); // 发送短信到指定电话号码
sub.setPhone(phone); // 添加电话号码到收信人实体
sublist.add(sub); // 将收信人实体短信添加到集合中
}
Dao.getDao().addSmsSendInfo(sendsms); // 保存短信发送信息到数据库
}
```

## 5.6 发信箱

### 5.6.1 功能概述

短信发送之后为了方便查看,本模块提供了发信箱的功能,它可以查看已经发送过的短信内容和该短信的接收人列表,可以避免重复发送相同的短信。发信箱的程序界面如图 5.25 所示。



图 5.25 发信箱界面

### 5.6.2 读取已发短信

发信箱从数据库中的“tb\_sendSms”、“tb\_sendSub”两个数据表中分别获取所发送短信的信息和所接收短信的电话列表。当用户单击主窗体左侧的“短信管理”选项卡中的“发信箱”列表项时,将触发列表组件的选择事件监听器去执行 noteManageListValueChanged()方法完成发送短信信息的读取。关键代码如下:





```

private void noteManageListViewValueChanged(
    javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        int index = noteManageList.getSelectedIndex();
        if (index == 0) {
            .....// 省略部分代码
        } else {
            notePanel.setTitle("发信箱");
            notePanel.setLxrPanelShow(true);
            List<TbSendsms> infos = Dao.getDao().getSmsSendInfo();
            notePanel.setSendSmsInfo(infos);
        }
        notePanel.revalidate();
        CardLayout layout = (CardLayout) contentPanel.getLayout();
        layout.show(contentPanel, "短信收发记录");
    }
}

```

上面的方法调用了 Dao 数据库操作类的 getSmsSendInfo() 方法获取发送的短信数据和收信人列表，这个方法的关键代码如下：

```

public List<TbSendsms> getSmsSendInfo() {
    List<TbSendsms> list = new ArrayList();
    try {
        String sql = "select * from tb_sendSms";
        ResultSet rs = conn.createStatement().executeQuery(sql);
        while (rs.next()) {
            TbSendsms sendsms = new TbSendsms();
            Collection<TbSendsub> subList = new ArrayList();
            sendsms.setTbSendsubCollection(subList);
            sendsms.setId(rs.getInt("id"));
            sendsms.setSmsdate(rs.getDate("smsdate"));
            sendsms.setSmsText(rs.getString("smstext"));
            sql = "select * from tb_sendSub where id=" + sendsms.getId();
            ResultSet subRs = conn.createStatement().executeQuery(sql);
            while (subRs.next()) {
                TbSendsub sub = new TbSendsub();
                sub.setCode(subRs.getInt("code"));
                sub.setId(sendsms.getId());
                sub.setPhone(subRs.getString("phone"));
                subList.add(sub);
            }
            list.add(sendsms);
        }
    } catch (SQLException ex) {
        Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);
    }
    return list;
}

```

在事件处理方法中调用了 notePanel (短信面板) 对象的 setSendSmsInfo() 方法，设置短信面板显示短信发送的信息，程序关键代码如下：

```

public void setSendSmsInfo(List<TbSendsms> infos) {
    // 创建表字段数组
    String[] columns = new String[]{"发送日期", "短信内容"};
    int size = infos.size();
    Object[][] data = new Object[size][2];
    for (int i = 0; i < size; i++) {
        TbSendsms send = infos.get(i);
        data[i][0] = send.getSmsdate();
        data[i][1] = send;
    }
    // 将短信实体存入 data 数组
    // 使用字段数组和数据数组创建表模型对象
    DefaultTableModel model = new DefaultTableModel(data, columns);
    smsTable.setModel(model);
}

```







```

smsTable.revalidate();
smsTable.getColumnModel().getColumn(0).setPreferredWidth(80);
smsTable.getColumnModel().getColumn(1).setPreferredWidth(600);
// 获取收信人列表组件的数据模型
ListSelectionModel selmodel = smsTable.getSelectionModel();
selmodel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
// 创建列表组件的选择事件监听器
listSelectionListenerImpl = new ListSelectionListenerImpl();
// 添加列表组件的选择事件监听器
selmodel.addListSelectionListener(listSelectionListenerImpl);
}

```

### 5.6.3 显示收信人列表

短信面板对象 notePanel 的 setSendSmsInfo() 方法在创建发送短信数据表模型的同时, 还为表格的选择模型添加了选择事件监听器, 这个监听器在用户选择表格上指定行的数据时, 负责执行选择事件的业务处理, 也就是读取与所选择的数据行相关的收信人列表信息, 然后显示到列表组件上。这个事件监听器的程序代码如下:

```

private class ListSelectionListenerImpl implements ListSelectionListener {
    public ListSelectionListenerImpl() { // 在构造方法中设置表格默认选择第一号
        if (smsTable.getRowCount() > 0) {
            smsTable.setRowSelectionInterval(0, 0);
            loadLxrList();
        }
    }
    public void loadLxrList() { // 读取收信人信息并添加到列表中
        int row = smsTable.getSelectedRow();
        if (row == -1) {
            return;
        }
        int index = smsTable.getColumnModel().getColumnIndex("短信内容");
        TbSendsms sendsms = (TbSendsms) smsTable.getValueAt(row, index);
        Collection<TbSendsub> phones = sendsms.getTbSendsubCollection();
        DefaultListModel model = new DefaultListModel();
        Iterator<TbSendsub> iterator = phones.iterator();
        while (iterator.hasNext()) {
            TbSendsub tbSendsub = iterator.next();
            model.addElement(tbSendsub);
        }
        lxrList.setModel(model);
    }
    public void valueChanged(ListSelectionEvent e) { // 选择事件的业务处理方法
        if (!e.getValueIsAdjusting()) {
            loadLxrList(); // 调用 loadLxrList() 方法装载收信人列表
        }
    }
}

```

## 5.7 联系人管理



139



### 5.7.1 功能概述

联系人管理是本模块的重要功能之一, 它将所有联系人的姓名、性别、电话、公司等



图 5.26 联系人管理界面

信息保存到数据库中，用户可以根据需要进行添加、删除和修改操作。当执行短信发送时，可以方便、快速地选择联系人。

联系人包含组别（即分类）信息，所以采用树组件显示，它可以使用父节点和子节点来区分组别和联系人。在树组件的上方有一个工具条，该工具条可以拖动以改变位置或浮动于窗体之上，工具栏中包括“添加”、“修改”、“删除”和“添加组别”4个管理按钮，如图 5.26 所示。

## 5.7.2 添加联系人组别

在添加联系人信息之前，必须创建联系人组别，也就是联系人的分类信息。单击工具栏上的“添加组别”按钮，将弹出如图 5.27 所示的对话框，要求输入联系人组别的名称。

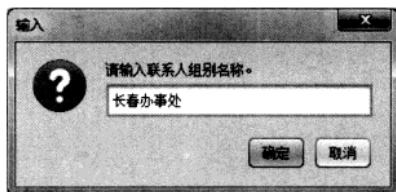


图 5.27 添加联系人组别

在用户输入联系人组别的名称并单击“确定”按钮之后，会将输入的联系人组别名称添加到数据库中。“添加组别”按钮的事件处理代码如下：

```
private void addLxrSuiteButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    // 弹出输入对话框，并获取输入的联系人组别名称  
    String name = JOptionPane.showInputDialog("请输入联系人组别名称。");  
    if (name == null || name.isEmpty()) { // 如果没有输入组别名称  
        return; // 终止本方法  
    }  
    Dao.getDao().addLxrSuite(name); // 添加联系人组别到数据库中  
    initLxrTree(); // 初始化联系人树组件  
}
```

在添加组别时，调用了 Dao 类的 addLxrSuite()方法，该方法负责保存组别名称到数据库中，程序关键代码如下：

```
public void addLxrSuite(String name) {  
    long id = System.currentTimeMillis(); // 创建数据表 ID  
    String sql = "insert into tb_lxrSuite values('" + id + "','" + name  
        + "')"; // 创建 SQL 语句  
    try {  
        conn.createStatement().execute(sql); // 执行 SQL 语句  
    } catch (SQLException ex) {  
        Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);  
    }  
}
```







将组别信息保存到数据库中之后, 必须调用 `initLxrTree()` 方法重新载入树组件的内容, 以更新界面, 保持与数据库的同步。在其他的添加联系人、修改联系人和删除联系人操作中都需要调用该方法更新树组件, 程序关键代码如下:

```
public void initLxrTree() {
    DefaultTreeModel model = (DefaultTreeModel) lxrTree.getModel();
    DefaultMutableTreeNode root = (DefaultMutableTreeNode) model.getRoot();
    root.removeAllChildren(); // 清除树组件的原始内容
    Dao dao = Dao.getDao(); // 初始化数据库操作类
    List lxrSuite = dao.getLxrSuite(); // 获取数据库中联系人记录
    if (lxrSuite == null) {
        return;
    }
    Iterator iterator = lxrSuite.iterator();
    while (iterator.hasNext()) { // 迭代遍历联系人记录
        String suite = (String) iterator.next(); // 获取组别信息
        // 使用组别名称创建树的父节点, 并允许有子节点
        DefaultMutableTreeNode node = new DefaultMutableTreeNode(suite, true);
        List lxrBeans = dao.getLxrBeans(suite); // 获取该组别的联系人信息
        Iterator lxrIterator = lxrBeans.iterator();
        while (lxrIterator.hasNext()) { // 使用联系人信息创建子节点
            TbLinkman lxr = (TbLinkman) lxrIterator.next();
            node.add(new DefaultMutableTreeNode(lxr, false));
        }
        root.add(node);
    }
    model.reload(); // 重新装载树模型
    lxrTree.setSelectionRow(0);
}
```

### 5.7.3 联系人对话框

联系人信息和组别信息不同, 它包含更多的信息, 除了名称之外, 还有性别、电话号码、组别、公司等。这就需要编写一个复杂一点的、能够接收所有联系人信息的对话框。如图 5.28 所示, 这个对话框将在添加和修改联系人时被调用。

如果单击对话框中的“更多信息”链接, 将显示更详细的输入文本框, 如图 5.29 所示。

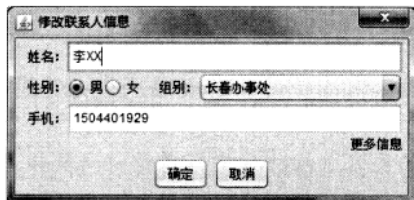


图 5.28 “修改联系人信息”对话框

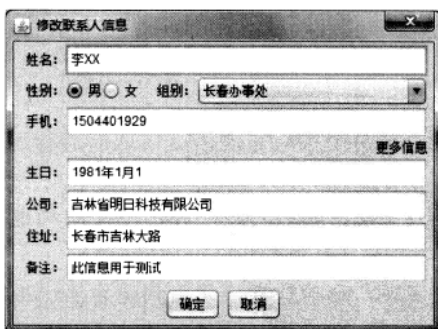


图 5.29 完整的联系人信息

#### 1. 定制构造方法

在本模块中的联系人对话框由 `LXRDialog` 类定义, 该类继承了 `javax.swing.JDialog` 类





成为对话框组件，它定义了两个构造方法，分别用于创建“添加联系人信息”对话框和“修改联系人信息”对话框。

### 1) 构造方法一

第一个构造方法用于创建标题为“添加联系人信息”的对话框，它简单地执行父类的构造方法，并调用 `initComponents()` 方法初始化对话框界面，最后设置对话框显示在屏幕居中的位置，程序关键代码如下：

```
public LXRDialog(NoteFrame parent, boolean modal) {
    super(parent, modal);           // 执行父类构造方法
    this.frame = parent;           // 保存父窗体的引用
    setTitle("添加联系人信息");     // 设置对话框标题
    initComponents();              // 初始化对话框界面
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    Dimension screenSize = toolkit.getScreenSize(); // 获取屏幕大小
    setLocation((screenSize.width - getWidth()) / 2, // 设置窗体居中显示
                (screenSize.height - getHeight()) / 2);
}
```

### 2) 构造方法二

第二个构造方法用于创建标题为“修改联系人信息”的对话框，这个构造方法多定义了一个 `linkman` 联系人参数，它是联系人实体对象，用于初始化界面中各个文本框内容。由于创建对话框界面和设置对话框位置等操作都已经在第一个构造方法中完成，所以这里直接调用第一个构造方法。然后修改对话框标题，并根据联系人实体对象修改界面文本框中的文本内容。程序关键代码如下：

```
public LXRDialog(TbLinkman linkman, NoteFrame parent, boolean modal) {
    this(parent, modal);           // 调用构造方法一
    this.linkman = linkman;        // 保存联系人实体的引用
    setTitle("修改联系人信息");     // 设置对话框标题
    nameTextField.setText(linkman.getName()); // 初始化联系人文本框
    String sex = linkman.getSex();
    if (sex.equals("男")) {         // 初始化性别选项
        sexGroup.setSelected(nanRadioButton.getModel(), true);
    } else {
        sexGroup.setSelected(nvRadioButton.getModel(), true);
    }
    phoneTextField.setText(linkman.getPhonenum()); // 初始化“手机”文本框
    suiteComboBox.setSelectedItem(linkman.getSuite()); // 初始化“组别”下拉选择框
    addressTextField.setText(linkman.getAddress()); // 初始化“住址”文本框
    companyTextField.setText(linkman.getCompany()); // 初始化“公司”文本框
    remarkTextField.setText(linkman.getRemark()); // 初始化“备注”文本框
    birthdayTextField.setText(linkman.getBirthday()); // 初始化“生日”文本框
}
```

有了这两个构造方法，就可以有选择地创建、添加或修改联系人的对话框了。



在调用同一个类中不同的构造方法时，可以使用 `this()` 语法并填充不同的构



142



造参数实现，而不是直接使用构造方法的名称。如果多个构造方法存在相同的操作，最好把它们放到一个默认的构造方法中再进行调用，这样能够提高代码重用，还可以减低产生错误的几率。

### 2. 初始化“组别”下拉选择框

对话框的“组别”下拉选择框中的内容是根据数据库中存储的数据而动态确定的，所以要在适当的位置读取数据库的组别信息，并填充到该下拉选择框中。这里通过对话框的





窗体打开事件完成组件的初始化操作。程序关键代码如下：

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {
    List lxrSuite = Dao.getDao().getLxrSuite(); // 获取数据库的联系人组别数据
    {
        // 初始化联系人组别下拉选择框
        DefaultComboBoxModel model = (DefaultComboBoxModel) suiteComboBox
            .getModel(); // 获取组件模型
        model.removeAllElements(); // 移除所有组件元素
        if (lxrSuite == null) {
            return;
        }
        Iterator iterator = lxrSuite.iterator();
        while (iterator.hasNext()) { // 遍历联系人组别的集合
            Object item = iterator.next();
            model.addElement(item); // 添加每个组别信息到组件中
            if (linkman != null && item.equals(linkman.getSuite())) {
                model.setSelectedItem(item); // 选择指定的联系人组别选项
            }
        }
    }
}
```



**说明** 这个组件的初始化步骤也可以在构造方法中完成，但那不是最理想的，特别

是在数据库应用中。试想一下，构造方法初始化了组件内容，但是在没有执行完该窗体的操作时，需要临时转到其他窗体完成修改数据库的操作，而这个操作可能会影响该窗体组件的内容。这时再回到该窗体中，组件的内容并不会随数据库的数据而改变，因为它只有在创建窗体时能被构造方法初始化。但是在窗体相应的事件监听器中（如窗体激活事件）执行组件的初始化操作，完全可以避免此类问题。

### 3. “确定”按钮的事件处理

单击联系人对话框的“确定”按钮，将执行联系人信息的保存工作，它首先获取界面中用户输入的所有信息，然后对关键的信息进行验证，以确保输入了主要的数据。最后，创建联系人实体对象，并保存到数据库中。程序关键代码如下：

```
private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String name = nameTextField.getText(); // 获取联系人姓名
    String sex = "男";
    if (nvRadioButton.isSelected()) { // 获取性别信息
        sex = "女";
    }
    String phone = phoneTextField.getText(); // 获取手机信息
    Object item = suiteComboBox.getSelectedItem();
    if (item == null) {
        return;
    }
    String suite = item.toString(); // 获取组别信息
    String address = addressTextField.getText(); // 获取住址信息
    String company = companyTextField.getText(); // 获取公司名称
    String remark = remarkTextField.getText(); // 获取备注信息
    String birthday = birthdayTextField.getText(); // 获取生日信息
    if (name == null || name.isEmpty()) { // 判断姓名是否为空
        infoLabel.setText("请填写联系人姓名!!!");
        infoLabel.setVisible(true);
        pack();
        return;
    } else if (phone == null || phone.isEmpty()) { // 判断手机号码是否为空
        infoLabel.setText("请填写联系人的手机号码!!!");
        infoLabel.setVisible(true);
    }
}
```







```
        pack();
        return;
    } else if (suite == null || suite.isEmpty()) { // 判断组别是否为空
        infoLabel.setText("请选择联系人组别!!! ");
        infoLabel.setVisible(true);
        pack();
        return;
    }
    TbLinkman lxr = new TbLinkman(); // 创建联系人实体对象
    if (linkman != null && linkman.getId() != null) { // 设置 ID 编号
        lxr.setId(linkman.getId());
    }
    lxr.setAddress(address); // 初始化联系人实体对象
    lxr.setBirthday(birthday);
    lxr.setCompany(company);
    lxr.setName(name);
    lxr.setPhonenum(phone);
    lxr.setRemark(remark);
    lxr.setSex(sex);
    lxr.setSuite(suite);
    Dao.getDao().saveOrUpdateLxr(lxr); // 保存或更新联系人实体到数据库
    frame.initLxrTree(); // 重新初始化联系人树组件
    dispose(); // 销毁对话框
}
```

在“确定”按钮的事件处理代码中，调用了 Dao 类的 saveOrUpdateLxr() 方法实现联系人实体的保存或更新操作。对于联系人对话框来说，这是非常重要的方法，该方法可以根据传入的联系人实体参数，判断是插入新的联系人数据，还是修改原有的联系人数据。程序关键代码如下：

```
public void saveOrUpdateLxr(TbLinkman lxr) {
    int id = getMaxId("tb_linkman", "id"); // 获取参数中的 ID 编号
    String sql;
    // 根据 id 编号判断生成插入语句还是生成更新语句
    if (lxr.getId() == null) {
        sql = "insert into tb_linkman values(?,?,?,?,?,?,?,?)";
    } else {
        id = lxr.getId();
        sql = "update tb_linkman set id=?,name=?,sex=?,birthday=?,"
            + "phonenum=?,address=?,company=?,suite=?,remark=?"
            + " where id=" + id;
    }
    try {
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setInt(1, id); // 设置 SQL 语句的参数
        ps.setString(2, lxr.getName());
        ps.setString(3, lxr.getSex());
        ps.setString(4, lxr.getBirthday());
        ps.setString(5, lxr.getPhonenum());
        ps.setString(6, lxr.getAddress());
        ps.setString(7, lxr.getCompany());
        ps.setString(8, lxr.getSuite());
        ps.setString(9, lxr.getRemark());
        ps.execute(); // 执行 SQL 语句
    } catch (SQLException ex) {
        Logger.getLogger(Dao.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```







### 5.7.4 添加联系人

单击工具栏上的“添加”按钮，将弹出“添加联系人信息”对话框，该对话框用于输入联系人信息，具体的对话框定义已经介绍过了。“添加”按钮需要做的就是事件监听器中，使用联系人对话框的第一个构造方法，创建添加联系人信息的对话框，接收用户的输入并保存到数据库。程序关键代码如下：

```
private void addLxrButtonActionPerformed(java.awt.event.ActionEvent evt) {
    LXRDialog dialog = new LXRDialog(this, true);    // 创建“添加联系人信息”对话框
    dialog.setVisible(true);                        // 设置对话框可见
}
```

### 5.7.5 修改联系人或组别

与添加联系人的按钮不同，“修改”按钮提供了修改联系人和修改组别的功能，它必须判断联系人树组件的当前选择相是父节点还是子节点，如果是父节点则执行修改组别名称的业务，如果是子节点则执行修改联系人信息的业务。程序关键代码如下：

```
private void modiLxrButtonActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) lxrTree
        .getSelectionPath().getLastPathComponent(); // 获取树组件被选择的节点
    Object value = node.getUserObject();           // 获取节点中的用户数据
    if (!node.getAllowsChildren()) {               // 如果选择的是子节点
        TbLinkman linkman = (TbLinkman) value;    // 转换用户数据为联系人实体类型
        // 创建“修改联系人信息”对话框
        LXRDialog dialog = new LXRDialog(linkman, this, true);
        dialog.setVisible(true);                   // 显示对话框
    } else if (!node.isRoot()) {                  // 如果选择的是父节点
        String suite = (String) value;             // 转换用户数据为字符串类型
        String type = JOptionPane.showInputDialog(this,
            "请输入组别名称", suite);              // 接收用户输入的组别名称
        if (type == null || type.isEmpty()) {
            return;
        }
        Dao.getDao().updateLxrSuite(suite, type); // 更新数据库中该组别的名称
        initLxrTree();                           // 重新初始化联系人树组件
    }
}
```

### 5.7.6 删除联系人或组别

与“修改”按钮一样，“删除”按钮也同时支持删除联系人和删除组别的功能，它也需要判断联系人树组件的当前选择项是父节点还是子节点，如果选择的是父节点，就执行删除组别的功能，如果选择的是子节点，那么，将执行删除联系人的功能。程序关键代码如下：

```
private void dellLxrButtonActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) lxrTree
        .getSelectionPath().getLastPathComponent(); // 获取树组件被选择的节点
    Object value = node.getUserObject();           // 获取节点中的用户数据
    if (node.getAllowsChildren() && !node.isRoot()) { // 如果选择的是父节点
        Dao.getDao().delLxrSuite(value.toString()); // 删除联系人组别
    }
}
```





```
}  
if (value instanceof TbLinkman) {           // 如果选择的是子节点  
    TbLinkman linkman = (TbLinkman) value;  
    Dao.getDao().delLxr(linkman);           // 删除联系人  
    node.removeFromParent();                // 删除相应的树节点  
}  
((DefaultTreeModel) lxrTree.getModel()).reload(); // 重新载入树组件模型  
}
```



由于短语管理功能与联系人管理功能的实现方法相似,这里不再重复介绍,请读者参见随书光盘中的源程序。





# 第 6 章

---

## FTP 上传下载模块

(Swing+FTP 技术实现)

FTP 协议是 Internet 中的文件传输通用协议，它位于 TCP/IP 协议的应用层，利用该协议，可以将一个完整的文件从一台计算机复制到另一台。但是在使用 FTP 传输文件前，需要先登录 FTP 服务器。用户可以通过注册的用户名和密码登录该服务器。如果软件允许，用户也可以匿名登录。本章设计了一个 FTP 客户端软件，主要实现了本地系统与远程 FTP 服务器间文件的上传与下载。通过本章的学习，读者能够学到：

- » 遍历 FTP 服务器目录
- » 获取本地文件的图标
- » 利用多线程实现 FTP 文件上传与下载
- » 本地文件与 FTP 文件的维护操作



## 6.1 FTP 上传下载模块概述

### 6.1.1 模块概述

FTP 上传下载模块是一个方便用户访问 FTP 服务器，执行常用操作的上传下载工具。它使用多线程技术同时完成文件的上传、下载和维护业务，因此不会耽误用户其他操作，不会导致 UI 界面死锁。此外，该模块的任务队列功能可以控制任务的先后顺序，暂停、继续和清空任务。

### 6.1.2 功能结构

FTP 文件管理模块包括连接 FTP 服务器、FTP 操作管理、本地操作管理、队列管理及帮助 5 部分，它的功能结构如图 6.1 所示。

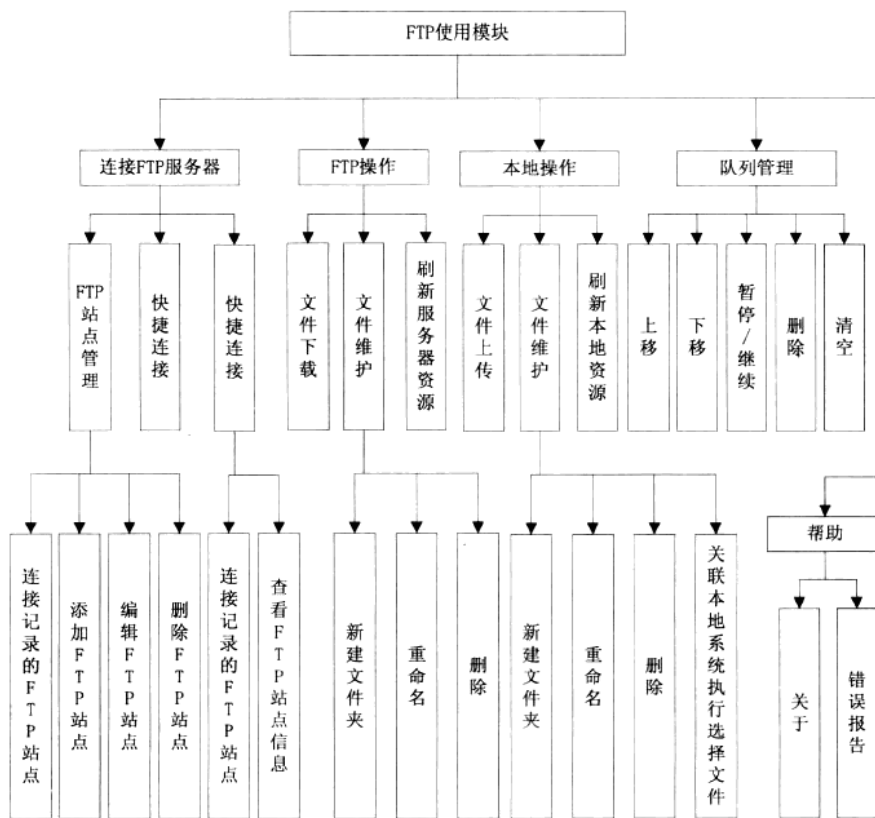


图 6.1 FTP 使用模块功能结构图



### 6.1.3 系统预览

本节将列出 FTP 使用模块的几个典型窗体, 其他窗体参见光盘中的源程序。系统主窗体的运行效果如图 6.2 所示, 主要的功能是管理整个 FTP 管理模块。

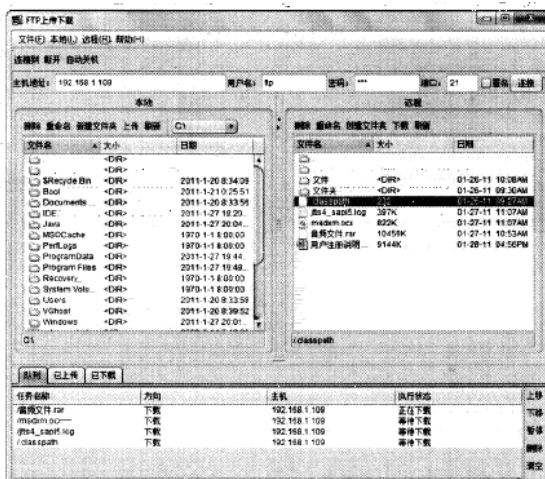


图 6.2 FTP 使用模块主界面

登录 FTP 服务器的面板包括连接、断开操作, 如图 6.3 所示。

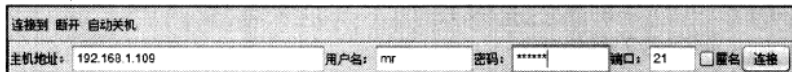


图 6.3 登录 FTP 服务器的面板

队列管理中记录已上传文件的程序界面如图 6.4 所示。

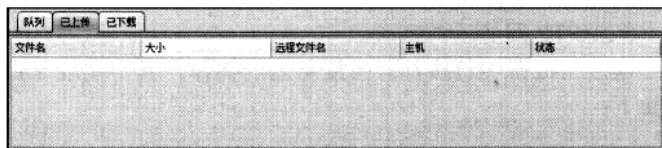


图 6.4 队列管理的文件上传记录

队列管理中记录上传、下载任务的程序界面如图 6.5 所示。

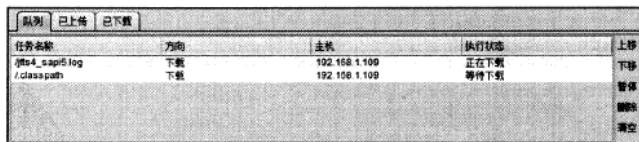


图 6.5 任务队列管理界面



本地资源管理界面如图 6.6 所示。

远程 FTP 资源管理界面如图 6.7 所示。

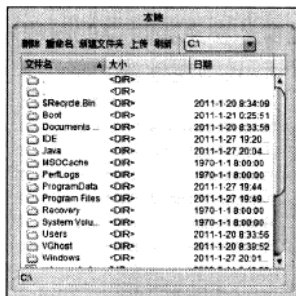


图 6.6 本地资源管理界面

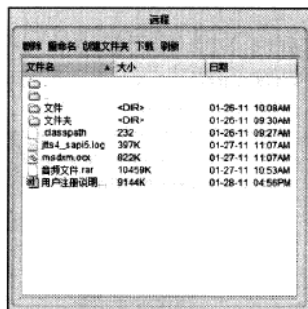


图 6.7 远程 FTP 资源管理界面

## 6.2 关键技术

### 6.2.1 登录 FTP 服务器

FTP 需要使用模块登录远程服务器，才能获取文件信息及进行其他操作。本模块使用 `FtpClient` 类中 `openServer()` 方法连接服务器，然后使用 `login()` 方法登录服务器。

`openServer()` 方法的声明如下：

```
openServer(String server, int port)
```

- `server`: 服务器的地址。
- `port`: 服务器的端口号。

`Login()` 方法的声明如下：

```
login(String userStr, String passStr)
```

- `userStr`: 登录 FTP 服务器的用户名。
- `passStr`: 登录 FTP 服务器的密码。

主窗体中的“连接”按钮负责执行登录服务器的操作，其中实现 FTP 服务器连接与登录的关键代码如下：

```
private void linkButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        String server = serverTextField.getText(); // 获取服务器地址  
        if (server == null) {  
            return;  
        }  
        String portStr = portTextField.getText(); // 获取端口号  
        if (portStr == null) {  
            portStr = "21";  
        }  
        int port = Integer.parseInt(portStr.trim());  
        userStr = userTextField.getText(); // 获取用户名  
        if (userStr.equals("") && checkboxes.isSelected()) {  
            userStr = "Anonymous";  
        } else {  
            userStr = userStr.trim();  
        }  
    }  
}
```





```

    }
    passStr = PassField.getText(); // 获取密码
    if (passStr.equals("") && checkboxes.isSelected()) {
        passStr = "331";
    } else {
        passStr = passStr.trim();
    }
    cutLinkButton.doClick();
    ftpClient = new FtpClient();
    ftpClient.openServer(server.trim(), port); // 连接服务器
    ftpClient.login(userStr, passStr); // 登录服务器
    ftpClient.binary(); // 使用二进制传输模式
    if (ftpClient.serverIsOpen()) { // 如果连接成功
        CUT_LINK_ACTION.setEnabled(true); // 设置断开按钮可用
    } else { // 否则
        CUT_LINK_ACTION.setEnabled(false); // 设置“断开”按钮不可用
        return; // 并结束事件处理
    }
    // 设置本地资源管理面板的 FTP 连接信息
    localPanel.setFtpClient(server, port, userStr, passStr);
    localPanel.getActionMap().get("uploadAction").setEnabled(true);
    // 设置“上传”按钮可用
    ftpPanel.setFtpClient(ftpClient); // 设置 FTP 资源管理面板的 FTP 连接信息
    // 设置“下载”按钮可用
    ftpPanel.getActionMap().get("downAction").setEnabled(true);
    ftpPanel.refreshCurrentFolder(); // 刷新 FTP 资源管理面板的当前文件夹
    queuePanel.startQueue(); // 启动任务队列线程
} catch (Exception ex) {
    ex.printStackTrace();
}
}

```

## 6.2.2 浏览本地资源

主窗体左侧是浏览本地文件信息的面板,它由一个滚动面板和一个表格控件组成。在表格控件中将显示本地资源信息。这些信息包括文件夹、文件的大小、名称、修改日期等。界面效果如图 6.8 所示。

将本地文件信息读取到表格的关键代码如下:



图 6.8 本地资源管理界面

```

private void listLocalFiles(File selDisk) {
    if (selDisk == null || selDisk.isFile()) {
        return;
    }
    localSelFilePathLabel.setText(selDisk.getAbsolutePath());
    File[] listFiles = selDisk.listFiles(); // 获取磁盘文件列表
    DefaultTableModel model = (DefaultTableModel) localDiskTable.getModel();
    // 获取表格的数据模型
    model.setRowCount(0); // 清除模型的内容
    model.addRow(new Object[] { ".", "<DIR>", "" }); // 创建.选项
    model.addRow(new Object[] { "..", "<DIR>", "" }); // 创建..选项
    if (listFiles == null) {
        JOptionPane.showMessageDialog(this, "该磁盘无法访问");
        return;
    }
    // 遍历磁盘根文件夹的内容, 添加到表格中
}

```



151







```
for (File file : listFiles) {
    File diskFile = new DiskFile(file);           // 创建文件对象
    String length = file.length() + "B ";         // 获取文件大小
    if (file.length() > 1000 * 1000 * 1000) {      // 计算文件G单位
        length = file.length() / 1000000000 + "G ";
    }
    if (file.length() > 1000 * 1000) {             // 计算文件M单位
        length = file.length() / 1000000 + "M ";
    }
    if (file.length() > 1000) {                    // 计算文件K单位
        length = file.length() / 1000 + "K ";
    }
    if (file.isDirectory()) {                      // 显示文件夹标志
        length = "<DIR>";
    }
    // 获取文件的最后修改日期
    String modifDate = new Date(file.lastModified()).toLocaleString();
    if (!file.canRead()) {
        length = "未知";
        modifDate = "未知";
    }
    // 将单个文件的信息添加到表格的数据模型中
    model.addRow(new Object[] { diskFile, length, modifDate });
}
localDiskTable.clearSelection();                 // 取消表格的选择项
}
```

### 6.2.3 浏览服务器资源

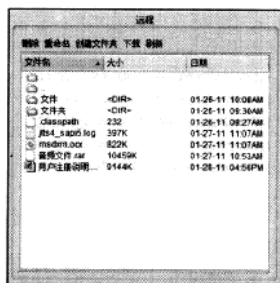


图 6.9 远程 FTP 资源管理界面

主窗体右侧的远程 FTP 文件信息面板显示的是服务器端的文件信息或文件夹信息。它显示文件信息的格式与本地资源界面相同，包括文件名、文件的大小、名称、修改日期等。界面效果如图 6.9 所示。

从 FTP 服务器读取文件目录信息到表格的关键步骤如下。

(1) 编写解析 FTP 文件信息格式的 listFtpFiles()方法，该方法要从服务器的返回信息中解析出每个文件的名称、大小、修改日期等，其关键代码如下：

```
public synchronized void listFtpFiles(final TelnetInputStream list) {
    // 获取表格的数据模型
    final DefaultTableModel model = (DefaultTableModel) ftpDiskTable.
        getModel();
    model.setRowCount(0);
    Runnable runnable = new Runnable() {         // 创建一个线程类
        @Override
        public synchronized void run() {
            ftpDiskTable.clearSelection();
            try {
                String pwd = getPwd();             // 获取 FTP 服务器的当前文件夹
                // 添加 "." 符号
                model.addRow(new Object[] { new FtpFile(".", pwd, true), "", "" });
                // 添加 ".." 符号
                model.addRow(new Object[] { new FtpFile("..", pwd, true), "", "" });
                BufferedReader br = new BufferedReader(new InputStreamReader
                    (list)); // 创建字符输入流
```





```
String data = null;
// 读取输入流中的文件目录
while ((data = br.readLine()) != null) {
    FtpFile ftpFile = new FtpFile(); // 创建 FTP 文件对象
    // 获取 FTP 服务器目录信息
    String dateStr = data.substring(0, 17).trim();
    String sizeOrDir = data.substring(18, 39).trim();
    String fileName = data.substring(39, data.length()).trim();
    // 将 FTP 目录信息初始化到 FTP 文件对象中
    ftpFile.setLastDate(dateStr);
    ftpFile.setSize(sizeOrDir);
    ftpFile.setName(fileName);
    ftpFile.setPath(pwd);
    model.addRow(new Object[] { ftpFile, ftpFile.getSize(),
    dateStr }); // 将文件信息添加到表格中
}
br.close(); // 关闭输入流
} catch (IOException ex) {
    Logger.getLogger(FTP_Client_Frame.class.getName()).log(Level.SEVERE, null, ex);
}
};
if (SwingUtilities.isEventDispatchThread()) // 启动线程对象
    runnable.run();
else
    SwingUtilities.invokeLater(runnable);
}
```

(2) 使用 `FtpClient` 类的 `list()` 方法获取 FTP 服务器的文件列表信息, 该方法将返回 `TelnetInputStream` 输入流, 然后使用 `listFtpFiles()` 方法来进行解析, 其关键代码如下:

```
public void refreshCurrentFolder() {
    try {
        TelnetInputStream list = ftpClient.list(); // 获取服务器文件列表
        listFtpFiles(list); // 调用解析方法
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

## 6.2.4 FTP 文件上传与下载

文件的上传和下载是本模块的核心功能, 下面将分别介绍其实现过程。

### 1. 文件上传

`sun.net.ftp.FtpClient` 类中提供的 `put()` 方法可以完成向 FTP 服务器传送文件的任务, 它返回一个 `TelnetOutputStream` 输出流。通过这个输出流可以向服务器发送文件数据, 最终将整个文件上传到服务器中。该方法的声明如下:

```
public TelnetOutputStream put(String file)
```

file: 上传到服务器的文件名称, 包括路径。

该方法在本模块的 `UploadThread` 上传线程类中被重复使用, 以完成文件上传任务, 其关键代码如下:

```
String remoteFile = path + "/" + file.getName(); // 远程 FTP 的文件名绝对路径
double fileLength = file.length() / Math.pow(1024, 2);
ProgressArg progressArg = new ProgressArg((int) (file.length() / 1024), 0, 0);
String size = String.format("%.4f MB", fileLength);
```





```
Object[] row = new Object[] { file.getAbsolutePath(), size, remoteFile,
ftpClient.getServer(), progressArg };
uploadPanel.addRow(row);
OutputStream put = ftpClient.put(remoteFile); // 获取服务器文件的输出流
FileInputStream fis = null; // 本地文件的输入流
try {
    fis = new FileInputStream(file); // 初始化文件的输入流
} catch (Exception e) {
    e.printStackTrace();
    return;
}
int readNum = 0;
byte[] data = new byte[1024]; // 缓存大小
while ((readNum = fis.read(data)) > 0) { // 读取本地文件到缓存
    Thread.sleep(0, 30); // 线程休眠
    put.write(data, 0, readNum); // 输出到服务器
    progressArg.setValue(progressArg.getValue() + 1); // 累加进度条
}
progressArg.setValue(progressArg.getMax()); // 结束进度条
fis.close(); // 关闭文件输入流
put.close(); // 关闭服务器输出流
```

## 2. 文件下载

文件下载是从服务器获取文件输入流，然后从这个输入流中获取文件内容到缓存，再写到本地文件中，最终完成文件的下载过程。sun.net.ftp.FtpClient 类中提供的 get() 方法可以获取远程 FTP 服务器上的文件输入流。该方法声明如下：

```
public TelnetInputStream get(String file)
```

file: 远程 FTP 服务器上的文件名称，这个文件名称包含路径信息。

该方法在本模块的 DownThread 下载线程类中被重复调用，以完成文件的下载任务，其关键代码如下：

```
// 获取服务器指定文件的输入流
TelnetInputStream ftpIs = ftpClient.get(file.getName());
if (ftpIs == null) {
    JOptionPane.showMessageDialog(this.ftpPanel, file.getName() + "无法下载");
    return;
}
File downFile = new File(localFolder, ftpFileStr); // 创建本地文件对象
// 创建本地文件的输出流
FileOutputStream fout = new FileOutputStream(downFile, true);
double fileLength = file.getLongSize() / Math.pow(1024, 2);
// 计算文件大小
ProgressArg progressArg = new ProgressArg((int) (file.getLongSize() / 1024), 0, 0);
String size = String.format("%.4f MB", fileLength);
Object[] row = new Object[] { ftpFileStr, size, downFile.getAbsolutePath(),
ftpClient.getServer(), progressArg };
DownloadPanel downloadPanel = ftpPanel.frame.getDownloadPanel();
downloadPanel.addRow(row);
byte[] data = new byte[1024]; // 定义缓存
int read = -1;
while ((read = ftpIs.read(data)) > 0) { // 读取 FTP 文件内容到缓存
    Thread.sleep(0, 30); // 线程休眠
    fout.write(data, 0, read); // 将缓存数据写入本地文件
    progressArg.setValue(progressArg.getValue() + 1); // 累加进度条
}
progressArg.setValue(progressArg.getMax()); // 结束进度条
fout.close(); // 关闭文件输出流
ftpIs.close(); // 关闭 FTP 文件输入流
```







## 6.2.5 向 FTP 服务器发送命令

FTP 使用模块中很多命令没有专门的方法可以调用, 需要向服务器发送 FTP 协议命令来完成操作, 例如, 创建文件夹、删除文件等。

sun.net.ftp.FtpClient 类中提供的 sendServer()方法可以向 FTP 服务器发送命令, 其声明如下:

```
public void sendServer(String command)
```

command: 向服务器发送的 FTP 协议的命令, 它是一个字符串文本。



**注意** 在调用 sendServer()方法时, 组成命令的字符串一定要以“\r\n”结尾, 否则

FTP 不会执行该命令。

FTP 协议有很多命令, 例如创建文件夹、重命名、改变文件夹、删除和退出等命令。这些命令的说明如表 6.1 所示。

表 6.1 FTP 协议的命令及说明

命 令	说 明
ABOR	中断数据连接
ACCT <account>	系统特权账号
ALLO <bytes>	为服务器上的文件存储器分配字节
APPE <filename>	追加内容到服务器同名文件
CDUP <dir path>	改变到服务器的上级目录
CWD <dir path>	改变服务器上的工作目录
DELE <filename>	删除服务器上的指定文件
HELP <command>	返回指定命令信息
LIST <name>	如果是文件则名列出文件信息, 如果是目录则列出文件列表
MODE <mode>	传输模式 (S=流模式, B=块模式, C=压缩模式)
MKD <directory>	在服务器上建立指定目录
NLST <directory>	列出指定目录内容
NOOP	无动作, 除了来自服务器上的许可信息
PASS <password>	系统登录密码
PASV	请求服务器等待数据连接
PORT <address>	IP 地址和两字节的端口 ID
PWD	显示当前工作目录
QUIT	从 FTP 服务器上退出登录
REIN	重新初始化登录状态连接





续表

命 令	说 明
REST <offset>	由特定偏移量重启文件传递
RETR <filename>	从服务器上找回（复制）文件
RMD <directory>	在服务器上删除指定目录
RNFR <old path>	对旧路径重命名
RNTO <new path>	对新路径重命名
SITE <params>	由服务器提供的站点特殊参数
SMNT <pathname>	挂载指定文件结构
STAT <directory>	在当前程序或目录上返回信息
STOR <filename>	存储（复制）文件到服务器上
STOU <filename>	存储文件到服务器名称上
STRU <type>	数据结构（F=文件，R=记录，P=页面）
SYST	返回服务器使用的操作系统
TYPE <data type>	数据类型（A=ASCII，E=EBCDIC，I=binary）
USER <username>>	系统登录的用户名

使用 `sendServer()` 方法向服务器发送命令后，需要使用 `readServerResponse()` 方法读取服务器返回代码，这个代码表明了服务器执行命令的状态。`readServerResponse()` 方法返回的 FTP 协议命令说明如表 6.2 所示。

表 6.2 FTP 协议的命令返回代码及说明

成 员	说 明	成 员	说 明
110	新文件指示器上的重启标记	331	要求密码
120	服务器准备就绪的分钟计数	332	要求账号
125	打开数据连接，开始传输	350	文件行为暂停
150	打开连接	421	服务关闭
200	成功	425	无法打开数据连接
202	命令没有执行	426	结束连接
211	系统状态回复	450	文件不可用
212	目录状态回复	451	遇到本地错误
213	文件状态回复	452	磁盘空间不足
214	帮助信息回复	500	无效命令
215	系统类型回复	501	错误参数
220	服务就绪	502	命令没有执行
221	退出网络	503	错误指令序列
225	打开数据连接	504	无效命令参数
226	结束数据连接	530	未登录网络
227	进入被动模式	532	存储文件需要账号
230	登录因特网	550	文件不可用
250	文件行为完成	551	不知道的页类型
257	路径名建立完成	552	超过存储分配

下面是本模块 FTP 文件删除操作的处理方法，它使用 `sendServer()` 方法向服务器传送了删除文件的命令，其关键代码如下：





```
private void delFile(FtpFile file) {
    FtpClient ftpClient = ftpPanel.ftpClient;        // 获取 ftpClient 实例
    try {
        if (file.isFile()) {                          // 如果删除的是文件
            // 发送删除文件的命令
            ftpClient.sendServer("DELE " + file.getName() + "\r\n");
            ftpClient.readServerResponse();           // 接收返回编码
        } else if (file.isDirectory()) {              // 如果删除的是文件夹
            ftpClient.cd(file.getName());             // 进入到该文件夹
            // 读取文件列表
            InputStreamReader list = new InputStreamReader(ftpClient.list());
            BufferedReader br = new BufferedReader(list);
            String nameStr = null;
            while ((nameStr = br.readLine()) != null) { // 解析每个文件
                Thread.sleep(0, 100);                // 线程休眠
                String name = nameStr.substring(39);   // 解析文件名
                String size = nameStr.substring(18, 39); // 解析文件大小
                FtpFile ftpFile = new FtpFile();       // 创建文件对象
                ftpFile.setName(name);                 // 设置文件名
                ftpFile.setPath(file.getAbsolutePath()); // 设置文件路径
                ftpFile.setSize(size);                 // 设置文件大小
                delFile(ftpFile);                      // 递归删除文件或文件夹
            }
            list.close();                             // 关闭读取文件列表的输入流
            br.close();
            ftpClient.cdUp();                          // 返回上层文件夹
            // 发送删除文件夹指令
            ftpClient.sendServer("RMD " + file.getName() + "\r\n");
            ftpClient.readServerResponse();           // 接收返回码
        }
    } catch (Exception ex) {
        Logger.getLogger(LocalPanel.class.getName()).log(Level.SEVERE,
            null, ex);
    }
}
```

### 6.2.6 获取文件在本系统的显示图标

主窗体本地资源和服务器资源界面中, 都使用了与操作系统相关联的图标。这样可以方便区分不同类型的文件, 例如图片文件、音乐文件等, 其运行效果如图 6.10 所示。

FileSystemView 类的 `getSystemIcon()` 方法可以根据 File 类型对象获取它在系统中显示的图标, 该方法的声明如下:

文件名	大小	日期
ProgramData	<DIR>	2011-1-27 19:44...
Program Files	<DIR>	2011-1-27 19:49...
Recovery	<DIR>	1970-1-1 8:00:00
System Volu...	<DIR>	1970-1-1 8:00:00
Users	<DIR>	2011-1-20 8:33:56
VGhost	<DIR>	2011-1-20 8:39:52
Windows	<DIR>	2011-1-27 20:01...
autoexec.bat	24B	2009-6-11 5:42:20
bootmgr	383K	2009-7-14 9:38:58
BOOTSECT...	8K	2011-1-21 0:25:53
config.sys	10B	2009-6-11 5:42:20
gridr	171K	2009-7-30 16:21...
hiberfil.sys	未知	未知
pagefile.sys	未知	未知
vt	179K	2007-12-10 17:2...
vt.mbr	8K	2007-11-6 16:19...

图 6.10 界面中显示文件的系统图标效果

```
public Icon getSystemIcon(File f)
```

f: 要获取图标的文件对象。

本模块为本地资源和服务器资源中的表格控件定制了渲染器 `FTPTableCellRenderer`, 其主要用途是使用 `FileSystemView` 类的 `getSystemIcon()` 方法返回值绘制每行记录的图标, 该类的代码如下:

```
public class FTPTableCellRenderer extends DefaultTableCellRenderer {
    private static final long serialVersionUID = -1750811013317754117L;
    // 文件夹图标
```







```
private final ImageIcon folderIcon = new
ImageIcon(getClass().getResource("/com/lzw/ftp/res/folderIcon.JPG"));
// 文件图标
private final ImageIcon fileIcon = new
ImageIcon(getClass().getResource("/com/lzw/ftp/res/fileIcon.JPG"));
private static FTPTableCellRenderer instance = null;
// 渲染器的实例对象
private FTPTableCellRenderer() {
}
public static FTPTableCellRenderer getCellRanderer() {
    if (instance == null)
        instance = new FTPTableCellRenderer();
    return instance;
}
@Override
protected void setValue(Object value) {
    if (value instanceof FileInterface) {
        FileInterface file = (FileInterface) value;
        // 获取 FileSystemView 类的实例对象
        FileSystemView view = FileSystemView.getFileSystemView();
        if (file.isDirectory()) {
            setText(file.toString());
            setIcon(folderIcon);
        } else {
            if (file instanceof File) { // 如果数据为 File 类
                Icon icon = view.getSystemIcon((File) file); // 获取文件的图标
                setIcon(icon); // 设置表格单元图标
            } else if (file instanceof FtpFile) { // 如果数据为 FtpFile 类
                FtpFile ftpfile = (FtpFile) file;
                try {
                    // 使用 FtpFile 的文件名称创建临时文件
                    File tempFile = File.createTempFile("tempfile_",
ftpfile.getName());
                    // 获取临时文件的图标
                    Icon icon = view.getSystemIcon(tempFile);
                    tempFile.delete(); // 删除临时文件
                    setIcon(icon); // 设置表格单元图标
                } catch (IOException e) {
                    e.printStackTrace();
                    setIcon(fileIcon);
                }
            }
            setText(file.toString()); // 设置文本内容
        }
    } else {
        setIcon(folderIcon); // 如果选择的不是文件或文件夹
        setText(value.toString()); // 设置备用的文件夹图标
        // 设置名称
    }
}
```

## 6.2.7 任务队列

本模块的任务队列功能可以记录上传/下载任务顺序，并包括显示任务执行状态、控制队列播放、改变任务顺序等功能。任务队列的运行效果如图 6.11 所示。

队列	已上传	已下载			
任务名称	方向	主机	执行状态	上传	下载
file4_sapif5.log	下载	192.168.1.109	正在下载		
/classpath	下载	192.168.1.109	等待下载		
				暂停	
				删除	
				清空	

图 6.11 任务队列管理界面



158







为了实现队列功能,需要使用 FIFO(先进先出)集合类对象。java.util 包中的 LinkedList 类是理想的选择,它实现了 Queue 接口中定义的全部方法,并且允许存放各种类型的元素。此外,该类还为列表开头和结尾的 get()、remove()和 insert()等操作提供统一的命名方法。下面介绍本模块使用的 LinkedList 类中的几个方法。

### 1. 添加队列元素

任务队列使用 FIFO 机制,所以新添加的任务需要保存在列表的末端,最后执行。LinkedList 类的 offer()方法将完成这一操作。该方法声明如下:

```
public boolean offer(E e)
```

e: 要添加到队列的元素。

在本模块的本地控制面板,包含“上传”按钮,在该按钮的事件处理方法中,就应用了 offer()方法,将表格中选择的所有文件分别添加到队列中,其关键代码如下:

```
public void actionPerformed(java.awt.event.ActionEvent evt) {
    int[] selRows = this.localPanel.localDiskTable.getSelectedRows();
    // 获取用户选择的多个文件或文件夹
    if (selRows.length < 1) {
        JOptionPane.showMessageDialog(this.localPanel,
            "请选择上传的文件或文件夹");
        return;
    }
    // 获取 FTP 服务器的当前路径
    String pwd = this.localPanel.frame.getFtpPanel().getPwd();
    FtpFile ftpFile = new FtpFile("", pwd, true); // 创建 FTP 当前路径的文件夹对象
    // 遍历本地资源的表格
    for (int i = 0; i < selRows.length; i++) {
        Object valueAt = this.localPanel.localDiskTable.getValueAt(selRows[i],
            0); // 获取表格选择行的第一列数据
        if (valueAt instanceof DiskFile) {
            final DiskFile file = (DiskFile) valueAt;
            // 获取本地面板类中的队列,该队列是 LinkedList 类的实例对象
            Queue<Object[]> queue = this.localPanel.queue;
            // 执行 offer 方法向队列尾添加对象
            queue.offer(new Object[] { file, ftpFile });
        }
    }
}
```

### 2. 查看队首元素

peek()方法可以获取和查看队首元素,但是并不把它从队列移除,该方法声明如下:

```
public E peek()
```



说明 如果队列为空,那么 peek()方法将返回 null 值。

本模块使用 peek()方法检验队首元素是不是上一次处理的任务,因为只是检测,所以不需要删除队列中元素,其关键代码如下:

```
public void run() { // 线程的主体方法
    while (conRun) {
        try {
            Thread.sleep(1000); // 线程休眠 1 秒
            Queue<Object[]> queue = localPanel.queue; // 获取本地面板的队列对象
            Object[] queueValues = queue.peek(); // 获取队首的对象
            if (queueValues == null) { // 如果该对象为空
                continue; // 进行下一次循环
            }
        }
    }
}
```





```
File file = (File) queueValues[0]; // 获取队列中的本地文件对象
FtpFile ftpFile = (FtpFile) queueValues[1]; // 获取队列中的 FTP 文件对象
if (file != null) {
    selPath = file.getParent();
    copyFile(file, ftpFile); // 调用递归方法上传文件
    FtpPanel ftpPanel = localPanel.frame.getFtpPanel();
    ftpPanel.refreshCurrentFolder(); // 刷新 FTP 面板中的资源
}
Object[] args = queue.peek();
// 判断队列顶是否为处理的上一个任务
if (queueValues == null || args == null || !queueValues[0].
equals(args[0])) {
    continue;
}
queue.remove(); // 移除队列首元素
} catch (Exception e) {
    e.printStackTrace();
}
}
```

### 3. 获取队首元素

获取队首元素的方法还有 `poll()` 方法。它与 `peek()` 方法不同，该方法将删除队首元素对象。该方法声明如下：

```
public E poll()
```

### 4. 移除队首元素

如果只是想简单地把队首元素移除，可以使用 `remove()` 方法，其声明如下：

```
public E remove()
```

## 6.3 FTP 站点管理

### 6.3.1 功能概述

FTP 站点管理功能，包括站点的连接、添加、编辑和删除功能，其运行效果如图 6.11 所示。

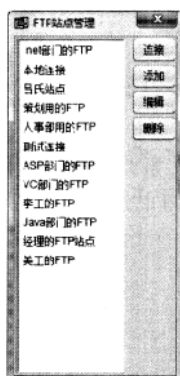


图 6.12 FTP 站点管理界面







### 6.3.2 读取属性文件

本模块使用 Properties 类将 FTP 站点信息以属性文件的格式保存在磁盘中,这样就实现了站点信息的持久化。程序在运行时需要使用 loadSiteProperties()方法重新加载属性文件,其关键代码如下:

```
private void loadSiteProperties() {
    try {
        if (!FILE.exists()) {
            FILE.getParentFile().mkdirs();
            FILE.createNewFile();
        }
        InputStream is = new FileInputStream(FILE);
        siteInfo.load(is);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

### 6.3.3 装载 FTP 站点信息

在 FTP 站点管理对话框的左侧是站点信息的列表,该列表显示了所有记录的站点名称,这个名称是用户自己定义的。这些名称是由列表控件数据模型中的站点信息对象提供的,其关键代码如下:

```
public void loadSiteList() {
    Enumeration<Object> keys = siteInfo.keys(); // 获取属性集合的键值集合
    DefaultListModel model = new DefaultListModel(); // 创建列表组件数据模型
    while (keys.hasMoreElements()) {
        String key = (String) keys.nextElement(); // 获取每个键值
        String value = siteInfo.getProperty(key); // 获取每个键值的内容
        // 使用键值和内容创建站点信息 Bean
        SiteInfoBean siteInfoBean = new SiteInfoBean(key, value);
        model.addElement(siteInfoBean); // 将站点信息对象添加到列表组件的模型中
    }
    list.setModel(model); // 设置列表组件使用创建的模型
}
```

编写 SiteInfoBean 类,它用于保存站点名称、服务器地址、登录用户名、FTP 服务器端口、ID 编号等信息。此外,该类还重写了 toString()方法,方便对象输出。其关键代码如下:

```
public class SiteInfoBean {
    private String siteName; // 站点名称
    private String server; // 服务器地址
    private String userName; // 登录用户名
    private int port; // FTP 服务器端口
    private String id; // ID 编号
    public SiteInfoBean(String siteName, String server, int port,
        String userName) {
        id = System.currentTimeMillis() + ""; // 赋值 ID 编号
        this.siteName = siteName; // 赋值服务器名称
        this.server = server; // 赋值服务器地址
    }
}
```





```
this.port = port;           // 赋值端口号
this.userName = userName;   // 赋值用户名
}
public SiteInfoBean(String id, String info) {
    this.id = id;           // 赋值 ID 编号
    String[] infos = info.split(","); // 解析属性信息字符串
    this.siteName = infos[0]; // 解析站点名称
    this.server = infos[1];   // 解析服务器地址
    this.port = Integer.valueOf(infos[2]); // 解析端口号
    this.userName = infos[3]; // 解析用户名
}
public String getSiteName() {
    return siteName;
}
public void setSiteName(String siteName) {
    this.siteName = siteName;
}
// 省略其他属性的 get 和 set 方法
@Override
public String toString() {
    return siteName;        // 这个返回值将显示在列表控件中
}
}
```

### 6.3.4 添加 FTP 站点

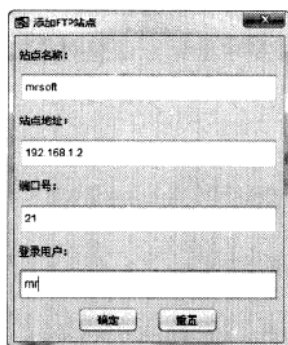


图 6.13 “添加 FTP 站点”对话框

在管理 FTP 站点时，首先要使用添加 FTP 站点的功能。添加的信息包括站点名称、站点地址、端口号、登录用户等，其运行效果如图 6.13 所示。

在 SiteDialog 类中定义了 3 个构造方法，分别用了完成 FTP 站点的添加、编辑和查看功能。这里着重讲解与添加相关的代码，对于其他两个功能，读者可以参考源代码文件自行学习。与添加功能相关的构造方法关键代码如下：

```
public SiteDialog(FtpSiteDialog frame) {
    super(frame);           // 调用父类的构造方法
    dialog = frame;         // 赋值父窗体对象
    initComponents();       // 调用初始化对话框界面的方法
}
```

SiteDialog 类实现了 ActionListener 接口，在该接口定义的 actionPerformed() 方法中，完成了对“确定”和“重置”两个按钮单击事件的监听，该方法的关键代码如下：

```
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand(); // 获取按钮的 command 属性
    if (command.equals("ok")) {           // 如果是“确定”按钮
        try {
            if (dialog == null) {
                dispose();
                return;
            }
        }
    }
}
```





```

// 获取界面所有文本框的内容
String siteName = siteNameField.getText().trim();
String server = siteAddressField.getText().trim();
String userName = loginUserField.getText().trim();
String portStr = portField.getText().trim();
// 判断是否填写了全部文本框
if (siteName.isEmpty() || server.isEmpty() || userName.isEmpty() ||
portStr.isEmpty()) {
    JOptionPane.showMessageDialog(this, "请填写全部信息");
    return;
}
int port = Integer.valueOf(portStr);
// 创建 FTP 站点信息的 JavaBean 对象
SiteInfoBean bean = new SiteInfoBean(siteName, server, port,
userName);
// 如果对话框的 siteBean 不为空
if (siteBean != null)
    bean.setId(siteBean.getId()); // 设置 FTP 站点的 ID 编号
dialog.addSite(bean); // 调用父窗体的 addSite 方法添加站点
dialog.loadSiteList(); // 调用父窗体的 loadSiteList 方法重载站点列表
dispose();
} catch (NullPointerException ex) {
    ex.printStackTrace();
    return;
} catch (NumberFormatException ex) {
    JOptionPane.showMessageDialog(this, "请正确填写端口号信息");
    ex.printStackTrace();
    return;
}
}
if (command.equals("cancel")) { // 如果是“重置”按钮
    if (siteBean == null) // 如果对话框的 siteBean 属性为空
        clearInput(); // 调用清除文本框内容的方法
    else // 否则
        initInput(); // 初始化界面文本框内容
}
}

```

## 6.4 本地资源管理

### 6.4.1 功能概述

本地资源管理面板位于主窗体左侧,用于获取本地磁盘中的文件和文件夹,并显示相关信息。在该面板的工具栏中还提供了“删除”、“重命名”、“新建文件夹”等工具按钮。此外,还有一个“上传”按钮,用于向 FTP 服务器传送文件,其运行效果如图 6.14 所示。



图 6.14 本地资源管理界面



说明 本节使用的类主要位于 com.lzw ftp.panel.local 包中。

## 6.4.2 删除本地文件

控制面板上的“删除”按钮用于删除表格中选择一个或多个文件（或文件夹），该功能由 DelFileAction 类完成。由于 File 类的 delete() 方法仅能删除文件和空文件夹，所以通常采用递归来实现删除非空文件夹的操作。DelFileAction 类的代码如下：

```
class DelFileAction extends AbstractAction {
    private static final long serialVersionUID = 2089609372999186478L;
    private LocalPanel localPanel; // 本地资源管理面板的引用对象
    public DelFileAction(LocalPanel localPanel, String name, Icon icon) {
        super(name, icon); // 调用父类的构造方法
        this.localPanel = localPanel; // 赋值本地资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获取表格选择的所有行
        final int[] selRows = this.localPanel.localDiskTable.getSelectedRows();
        if (selRows.length < 1) // 如果没有选择表格内容
            return; // 结束该方法
        // 用户确认是否删除
        int confirmDialog = JOptionPane.showConfirmDialog(localPanel,
            "确定要执行删除吗?");
        if (confirmDialog == JOptionPane.YES_OPTION) { // 如果用于同意删除
            Runnable runnable = new Runnable() { // 创建线程
                private void delFile(File file) {
                    try {
                        if (file.isFile()) { // 如果删除的是文件
                            boolean delete = file.delete(); // 调用删该文件的方法
                            if (!delete) {
                                JOptionPane.showMessageDialog(localPanel,
                                    file.getAbsolutePath() +
                                    "文件无法删除。", "删除文件",
                                    JOptionPane.ERROR_MESSAGE);
                                return;
                            }
                        } else if (file.isDirectory()) { // 如果删除的是文件夹
                            // 获取该文件夹的文件列表
                            File[] listFiles = file.listFiles();
                            if (listFiles.length > 0) {
                                for (File subFile : listFiles) {
                                    // 调用递归方法删除该列表的所有文件或文件夹
                                    delFile(subFile);
                                }
                            }
                            boolean delete = file.delete(); // 最后删除该文件夹
                            if (!delete) { // 如果成功删除
                                JOptionPane.showMessageDialog(localPanel,
                                    file.getAbsolutePath() +
                                    "文件夹无法删除。", "删除文件",
                                    JOptionPane.ERROR_MESSAGE);
                                return;
                            }
                        }
                    } catch (Exception ex) {
                        Logger.getLogger(LocalPanel.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            };
            runnable.run(); // 返回方法的调用处
        }
    }
}
```







```

    }
    @Override
    public void run() {
        File parent = null;
        // 遍历表格的选择内容
        for (int i = 0; i < selRows.length; i++) {
            // 获取每个选择行的第一列单元内容
            Object value = DelFileAction.this.localPanel.
localDiskTable.getValueAt(selRows[i], 0);
            // 如果该内容不是 DiskFile 类的实例对象
            if (!(value instanceof DiskFile))
                continue; // 结束本次循环
            DiskFile file = (DiskFile) value;
            if (parent == null)
                parent = file.getParentFile(); // 获取选择文件的上级文件夹
            if (file != null) {
                delFile(file); // 调用递归方法删除选择内容
            }
        }
        // 调用 refreshFolder 方法刷新当前文件夹
        DelFileAction.this.localPanel.refreshFolder(parent);
        JOptionPane.showMessageDialog(localPanel, "删除成功。");
    }
    new Thread(runnable).start(); // 创建并启动这个线程
}
}
}

```

### 6.4.3 重命名本地文件或文件夹

控制面板上的“重命名”按钮用于重命名表格中选择一个文件或文件夹，该功能由 RennameAction 类完成。通过 File 类的 renameTo() 方法可以对文件或文件夹进行重命名。RennameAction 类的代码如下：

```

class RennameAction extends AbstractAction {
    private static final long serialVersionUID = -2972330607581798822L;
    private LocalPanel localPanel; // 本地资源管理面板的引用
    public RennameAction(LocalPanel localPanel, String name, Icon icon) {
        super(name, icon); // 调用父类的构造方法
        this.localPanel = localPanel; // 赋值本地资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获取本地资源表格的选择行号
        int selRow = this.localPanel.localDiskTable.getSelectedRow();
        if (selRow < 0)
            return;
        // 获取选择行的第一个单元值
        Object value = this.localPanel.localDiskTable.getValueAt(selRow, 0);
        if (!(value instanceof File))
            return;
        File file = (File) value; // 将该单元值转换为 File 类的对象
        // 使用对话框接收用户如入的新文件名
        String fileName = JOptionPane.showInputDialog("请输入新文件名",
file.getName());
        if (fileName == null)
            return;
        // 创建新名称的文件
        File renFile = new File(file.getParentFile(), fileName);
        System.out.println(renFile);
    }
}

```







```
boolean isRename = file.renameTo(renFile); // 将原文件重命名
this.localPanel.refreshFolder(file.getParentFile()); // 刷新文件夹
if (isRename) {
    JOptionPane.showMessageDialog(this.localPanel,
        "重命名为" + fileName + "成功。");
} else {
    JOptionPane.showMessageDialog(this.localPanel,
        "无法重命名为" + fileName + "。", "文件重命名",
        JOptionPane.ERROR_MESSAGE);
}
}
```

## 6.4.4 新建文件夹

控制面板上的“新建文件夹”按钮用于新建文件夹，该功能由 `CreateFolderAction` 类完成。通过 `File` 类的 `mkdir()` 方法可以新建文件夹。`CreateFolderAction` 类的代码如下：

```
class CreateFolderAction extends AbstractAction {
    private static final long serialVersionUID = 8768984367943164978L;
    private LocalPanel localPanel; // 本地资源管理面板的引用
    public CreateFolderAction(LocalPanel localPanel, String name, Icon icon) {
        super(name, icon); // 调用父类构造方法
        this.localPanel = localPanel; // 赋值本地资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 使用输入对话框接收用户输入的文件夹名称
        String folderName = JOptionPane.showInputDialog("请输入文件夹名称：");
        if (folderName == null)
            return;
        File curFolder = null;
        int selRow = localPanel.localDiskTable.getSelectedRow();
        // 获取本地资源表格的当前选择行号
        if (selRow < 0) {
            curFolder = new File(localPanel.localSelFilePathLabel.getText());
            // 创建当前文件夹对象
        } else {
            // 获取表格选择行的第一个单元值
            Object value = localPanel.localDiskTable.getValueAt(selRow, 0);
            if (value instanceof File) { // 如果单元值是文件，则获取文件的上级文件夹
                curFolder = (File) value;
                if (curFolder.getParentFile() != null)
                    curFolder = curFolder.getParentFile();
            } else
                // 否则根据界面的路径标签创建当前文件夹对象
                curFolder = new File(localPanel.localSelFilePathLabel.getText());
            // 创建当前文件夹下的新文件夹对象
            File tempFile = new File(curFolder, folderName);
            if (tempFile.exists()) { // 如果存在相同文件或文件夹
                // 提示用户名称已存在
                JOptionPane.showMessageDialog(localPanel, folderName +
                    "创建失败，已经存在此名称的文件夹或文件。", "创建文件夹",
                    JOptionPane.ERROR_MESSAGE);
                return; // 结束本方法
            }
            if (tempFile.mkdir()) // 创建文件夹
                JOptionPane.showMessageDialog(localPanel, folderName + "文件夹，创建成功。", "创建文件夹", JOptionPane.INFORMATION_MESSAGE);
            else
                JOptionPane.showMessageDialog(localPanel, folderName + "文件夹无法被创建。", "创建文件夹", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```



```
this.localPanel.refreshFolder(curFolder); // 刷新文件夹
```

### 6.4.5 添加本地文件到上传队列

控制面板上的“上传”按钮用于上传文件到服务器，该功能由 UploadAction 类完成，它继承了 AbstractAction 类，并重写了事件处理方法。在该方法中获取了当前选择的所有文件或文件夹，然后创建 DiskFile 类和 FtpFile 类的对象，并将这些文件对象添加到本地面板队列中。UploadAction 类的代码如下：

```
class UploadAction extends AbstractAction {
    private static final long serialVersionUID = -6505104249434545886L;
    private LocalPanel localPanel; // 本地资源管理面板的引用
    public UploadAction(LocalPanel localPanel, String name, Icon icon) {
        super(name, icon); // 调用父类构造方法
        this.localPanel = localPanel; // 赋值本地资源管理面板的引用
        setEnabled(false); // 设置按钮不可用
    }
    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        int[] selRows = this.localPanel.localDiskTable.getSelectedRows();
        // 获取用户选择的多个文件或文件夹
        if (selRows.length < 1) {
            JOptionPane.showMessageDialog(this.localPanel, "请选择上传的文件或文件夹");
            return;
        }
        String pwd = this.localPanel.frame.getFtpPanel().getPwd();
        // 获取 FTP 服务器的当前路径
        FtpFile ftpFile = new FtpFile("", pwd, true); // 创建 FTP 当前路径的文件夹对象
        // 遍历本地资源的表格
        for (int i = 0; i < selRows.length; i++) {
            // 获取表格选择行的第一列数据
            Object valueAt = this.localPanel.localDiskTable.getValueAt(
                selRows[i], 0);
            if (valueAt instanceof DiskFile) {
                final DiskFile file = (DiskFile) valueAt;
                // 获取本地面板类中的队列，该队列是 LinkedList 类的实例对象
                Queue<Object> queue = this.localPanel.queue;
                // 执行 offer 方法向队列尾添加对象
                queue.offer(new Object[] { file, ftpFile });
            }
        }
    }
}
```

FtpFile 类是本模块自定义的一个 JavaBean，它用于保存 FTP 服务器的当前文件或文件夹信息，其关键代码如下：

```
public class FtpFile implements FileInterface {
    private String name = ""; // 文件名称
    private String path = ""; // 路径
    protected boolean directory; // 是否为文件夹
    private boolean file; // 是否为文件
    private String lastDate; // 最后修改日期
    private String size; // 文件大小
    private long longSize; // 文件大小的长整型类型
    private final int GB = (int) Math.pow(1024, 3); // GB 单位数值
    private final int MB = (int) Math.pow(1024, 2); // MB 单位数值
    private final int KB = 1024; // KB 单位数值
    public FtpFile() {
```







```
}
public FtpFile(String name, String path, boolean directory) {
    this.name = name;                                // 初始化相关属性
    this.path = path;
    this.directory = directory;
}
public String getSize() {
    return size;
}
public void setSize(String nsize) {
    if (nsize.indexOf("DIR") != -1) {                // 如果文件大小包含 DIR 字符串
        this.size = "<DIR>";
        directory = true;                            // 设置该类成为一个文件夹对象
        file = false;
    } else {                                         // 否则
        file = true;                                // 设置该类成为一个文件对象
        directory = false;
        this.size = nsize.trim();
        longSize = Long.parseLong(size);            // 计算文件的大小单位
        if (longSize > GB) {
            size = longSize / GB + "G ";
        }
        if (longSize > MB) {
            size = longSize / MB + "M ";
        }
        if (longSize > KB) {
            size = longSize / KB + "K ";
        }
    }
}
// 省略其他 get 和 set 方法
}
```

#### 6.4.6 刷新本地资源列表

控制面板上的“刷新”按钮用于重新加载本地磁盘中的文件和文件夹。在对本地文件或文件夹进行操作后，例如重命名文件、删除文件等，需要进行刷新，否则用户很难确定操作是否成功执行。刷新功能由 `RefreshAction` 类完成，它使用 `LocalPanel` 类的 `listLocalFiles()` 方法实现。`RefreshAction` 类的代码如下：

```
class RefreshAction extends AbstractAction {
    private static final long serialVersionUID = -2114228419376013892L;
    private LocalPanel localPanel;                    // 本地资源管理面板的引用
    public RefreshAction(LocalPanel localPanel, String name, Icon icon) {
        super(name, icon);                            // 执行父类的构造方法
        this.localPanel = localPanel;                  // 赋值本地资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        this.localPanel.refreshCurrentFolder();        // 调用管理面板的刷新方法
    }
}
```

`LocalPanel` 类的 `listLocalFiles()` 方法的关键代码如下：

```
private void listLocalFiles(File selDisk) {
    if (selDisk == null || selDisk.isFile()) {
        return;
    }
    localSelFilePathLabel.setText(selDisk.getAbsolutePath());
    File[] listFiles = selDisk.listFiles();           // 获取磁盘文件列表
    // 获取表格的数据模型
}
```







```

DefaultTableModel model = (DefaultTableModel) localDiskTable.getModel();
model.setRowCount(0); // 清除模型的内容
model.addRow(new Object[] { ".", "<DIR>", "" }); // 创建“.”选项
model.addRow(new Object[] { "..", "<DIR>", "" }); // 创建“..”选项
if (listFiles == null) {
    JOptionPane.showMessageDialog(this, "该磁盘无法访问");
    return;
}
// 遍历磁盘根文件夹的内容, 添加到表格中
for (File file : listFiles) {
    File diskFile = new DiskFile(file); // 创建文件对象
    String length = file.length() + "B "; // 获取文件大小
    if (file.length() > 1000 * 1000 * 1000) { // 计算文件 G 单位
        length = file.length() / 1000000000 + "G ";
    }
    if (file.length() > 1000 * 1000) { // 计算文件 M 单位
        length = file.length() / 1000000 + "M ";
    }
    if (file.length() > 1000) { // 计算文件 K 单位
        length = file.length() / 1000 + "K ";
    }
    if (file.isDirectory()) { // 显示文件夹标志
        length = "<DIR>";
    }
    String modifDate = new Date(file.lastModified()).toLocaleString();
    // 获取文件的最后修改日期
    if (!file.canRead()) {
        length = "未知";
        modifDate = "未知";
    }
    // 将单个文件的信息添加到表格的数据模型中
    model.addRow(new Object[] { diskFile, length, modifDate });
}
localDiskTable.clearSelection(); // 取消表格的选择项
}

```

## 6.5 FTP 资源管理

### 6.5.1 功能概述

FTP 资源管理面板位于主窗体右侧, 用于获取 FTP 服务器中的文件和文件夹, 并显示相关信息。在该面板的工具栏中, 还提供了“删除”、“重命名”、“创建文件夹”和“刷新”等工具按钮。此外, 还有一个“下载”按钮, 用于下载 FTP 服务器上的资源, 其运行效果如图 6.15 所示。

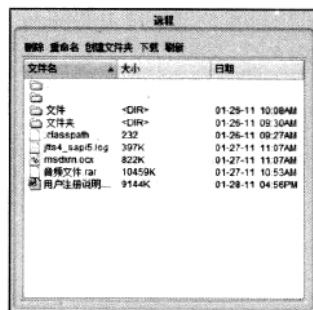


图 6.15 远程 FTP 资源管理界面







说明 本节使用的类主要位于 com.lzw.ftp.paneftp 包中。

## 6.5.2 删除服务器文件

控制面板上的“删除”按钮用于删除表格中选择一个或多个文件（或文件夹），该功能由 DelFileAction 类完成。由于 File 类的 delete() 方法仅能删除文件和空文件夹，所以通常采用递归来实现删除非空文件夹的操作。DelFileAction 类的代码如下：

```
class DelFileAction extends AbstractAction {
    private static final long serialVersionUID = -8490779915524602529L;
    private FtpPanel ftpPanel;
    public DelFileAction(FtpPanel ftpPanel, String name, Icon icon) {
        super(name, icon);
        this.ftpPanel = ftpPanel;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获取显示 FTP 资源列表的表格组件当前选择的所有行
        final int[] selRows = ftpPanel.ftpDiskTable.getSelectedRows();
        if (selRows.length < 1)
            return;
        int confirmDialog = JOptionPane.showConfirmDialog(ftpPanel,
            "确定要删除吗? ");
        if (confirmDialog == JOptionPane.YES_OPTION) {
            Runnable runnable = new Runnable() {
                private void delFile(FtpFile file) {
                    FtpClient ftpClient = ftpPanel.ftpClient; // 获取 ftpClient 实例
                    try {
                        if (file.isFile()) { // 如果删除的是文件
                            ftpClient.sendServer("DELE " + file.getName() + "\r\n");
                            // 发送删除文件的命令
                            ftpClient.readServerResponse(); // 接收返回编码
                        } else if (file.isDirectory()) { // 如果删除的是文件夹
                            ftpClient.cd(file.getName()); // 进入到该文件夹
                            InputStreamReader list = new InputStreamReader
                                (ftpClient.list()); // 读取文件列表
                            BufferedReader br = new BufferedReader(list);
                            String nameStr = null;
                            // 解析每个文件
                            while ((nameStr = br.readLine()) != null) {
                                Thread.sleep(0, 100); // 线程休眠
                                String name = nameStr.substring(39); // 解析文件名
                                // 解析文件大小
                                String size = nameStr.substring(18, 39);
                                FtpFile ftpFile = new FtpFile(); // 创建文件对象
                                ftpFile.setName(name); // 设置文件名
                                // 设置文件路径
                                ftpFile.setPath(file.getAbsolutePath());
                                ftpFile.setSize(size); // 设置文件大小
                                delFile(ftpFile); // 递归删除文件或文件夹
                            }
                            list.close(); // 关闭读取文件列表的输入流
                            br.close();
                            ftpClient.cdUp(); // 返回上层文件夹
                        } // 发送删除文件夹指令
                        ftpClient.sendServer("RMD " + file.getName() + "\r\n");
                        ftpClient.readServerResponse(); // 接收返回码
                    }
                }
            };
            runnable.run();
        }
    }
}
```







```

        } catch (Exception ex) {
            Logger.getLogger(LocalPanel.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    @Override
    public void run() {
        // 遍历显示 FTP 资源的表格的所有选择行
        for (int i = 0; i < selRows.length; i++) {
            // 获取每行的第一个单元值, 并转换为 FtpFile 类型
            final FtpFile file = (FtpFile) ftpPanel.ftpDiskTable
                .getValueAt(selRows[i], 0);
            if (file != null) {
                delFile(file); // 调用删除文件的递归方法
                try {
                    // 向服务器发送删除文件夹的方法
                    ftpPanel.ftpClient.sendServer("RMD " + file.getName() +
                        "\r\n");
                    // 读取 FTP 服务器的返回码
                    ftpPanel.ftpClient.readServerResponse();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        // 刷新 FTP 服务器资源列表
        DelFileAction.this.ftpPanel.refreshCurrentFolder();
        JOptionPane.showMessageDialog(ftpPanel, "删除成功.");
    }
}
};
new Thread(runnable).start();
}
}

```

### 6.5.3 重命名服务器文件或文件夹

控制面板上的“重命名”按钮用于重命名表格中选择一个文件或文件夹, 该功能由 `RenameAction` 类完成。通过 `File` 类的 `renameTo()` 方法可以对文件或文件夹进行重命名。`RenameAction` 类的代码如下:

```

class RenameAction extends AbstractAction {
    private static final long serialVersionUID = -8562738709144753730L;
    private FtpPanel ftpPanel;
    public RenameAction(FtpPanel ftpPanel, String name, Icon icon) {
        super(name, icon); // 调用父类的构造方法
        this.ftpPanel = ftpPanel; // 赋值 FTP 资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获取显示 FTP 资源的表格当前选择行号
        int selRow = ftpPanel.ftpDiskTable.getSelectedRow();
        if (selRow < 0)
            return;
        // 获取当前行的第一个表格单元值, 并转换成 FtpFile 类型的对象
        FtpFile file = (FtpFile) ftpPanel.ftpDiskTable.getValueAt(selRow, 0);
        // 使用对话框接收用户输入的新文件或文件夹名称
        String newName = JOptionPane.showInputDialog(ftpPanel, "请输入新名称.");
        if (file.getName().equals(".") || file.getName().equals(".."))
            || newName == null)
            return;
        try {

```







```
// 向服务器发送重命名的指令
ftpPanel.ftpClient.sendServer("RNFR " + file.getName() + "\r\n");
ftpPanel.ftpClient.readServerResponse();
ftpPanel.ftpClient.sendServer("RNT0 " + newName + "\r\n");
ftpPanel.ftpClient.readServerResponse();
ftpPanel.refreshCurrentFolder(); // 刷新当前文件夹
} catch (IOException el) {
    el.printStackTrace();
}
}
```

### 6.5.4 新建文件夹

控制面板上的“新建文件夹”按钮用于新建文件夹，该功能由 `CreateFolderAction` 类完成。通过 `File` 类的 `mkdir()` 方法可以新建文件夹。`CreateFolderAction` 类的代码如下：

```
class CreateFolderAction extends AbstractAction {
    private static final long serialVersionUID = -8728823346448114146L;
    private FtpPanel ftpPanel;
    public CreateFolderAction(FtpPanel ftpPanel, String name, Icon icon) {
        super(name, icon); // 调用父类构造方法
        this.ftpPanel = ftpPanel; // 赋值 FTP 资源管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // 接收用户输入的新建文件夹的名称
        String folderName = JOptionPane.showInputDialog("请输入文件夹名称: ");
        if (folderName == null)
            return;
        int read = -1;
        try {
            // 发送创建文件夹的命令
            ftpPanel.ftpClient.sendServer("MKD " + folderName + "\r\n");
            // 读取 FTP 服务器的命令返回码
            read = ftpPanel.ftpClient.readServerResponse();
        } catch (IOException el) {
            el.printStackTrace();
        }
        if (read == 257) // 如果返回码等于 257 (路径名建立完成)
            // 提示文件夹创建成功
            JOptionPane.showMessageDialog(ftpPanel, folderName +
                "文件夹, 创建成功。", "创建文件夹",
                JOptionPane.INFORMATION_MESSAGE);
        else
            // 否则提示用户该文件夹无法创建
            JOptionPane.showMessageDialog(ftpPanel, folderName +
                "文件夹无法被创建。", "创建文件夹",
                JOptionPane.ERROR_MESSAGE);
        this.ftpPanel.refreshCurrentFolder();
    }
}
```



### 6.5.5 添加服务器资源到下载队列

控制面板上的“下载”按钮用于从服务器下载文件，该功能由 `DownAction` 类完成，它继承了 `AbstractAction` 类，并重写了事件处理方法。在该方法中获取了当前选择的所有



文件或文件夹，然后创建 FtpFile 类和 File 类的对象，并将这些文件对象添加到下载队列中。DownAction 类的代码如下：

```
class DownAction extends AbstractAction {
    private static final long serialVersionUID = 5517561846481650190L;
    private final FtpPanel ftpPanel;
    public DownAction(FtpPanel ftpPanel, String name, Icon icon) {
        super(name, icon);           // 调用父类构造方法
        this.ftpPanel = ftpPanel;    // 赋值 FTP 资源管理面板的引用
        setEnabled(false);           // 设置动作不可用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        final int[] selRows = ftpPanel.ftpDiskTable.getSelectedRows();
        // 获取 FTP 资源表格的所有选择行
        if (selRows.length < 1)
            return;
        // 遍历表格的所有选择行
        for (int i = 0; i < selRows.length; i++) {
            // 获取每行的第一个单元值并转换成 FtpFile 类的对象
            final FtpFile file = (FtpFile) ftpPanel.ftpDiskTable
                .getValueAt(selRows[i], 0);
            if (file != null) {
                // 获取本地资源管理面板的当前文件夹
                File currentFolder = ftpPanel.frame.getLocalPanel()
                    .getCurrentFolder();
                // 把 FTP 文件对象和本地当前文件夹对象定义成数组添加到下载队列中
                ftpPanel.queue.offer(new Object[] { file, currentFolder });
            }
        }
    }
}
```

### 6.5.6 刷新服务器资源列表

控制面板上的“刷新”按钮用于重新加载服务器中的文件和文件夹。在对服务器文件或文件夹进行操作后，例如重命名文件、删除文件等，需要进行刷新，否则用户很难确定操作是否成功执行。刷新功能由 RefreshAction 类完成，它使用 FtpPanelPanel 类的 listFtpFiles() 方法实现。RefreshAction 类的代码如下：

```
class RefreshAction extends AbstractAction {
    private static final long serialVersionUID = -755226093654466353L;
    private FtpPanel ftpPanel;
    public RefreshAction(FtpPanel ftpPanel, String name, Icon icon) {
        super(name, icon);           // 调用父类构造方法
        this.ftpPanel = ftpPanel;    // 赋值 FTP 管理面板的引用
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        ftpPanel.refreshCurrentFolder(); // 调用刷新 FTP 资源列表的方法
    }
}
```

FtpPanelPanel 类的 refreshCurrentFolder () 方法代码如下：

```
public void refreshCurrentFolder() {
    try {
        TelnetInputStream list = ftpClient.list(); // 获取服务器文件列表
        listFtpFiles(list);                       // 调用解析方法
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```
}  
}
```

FtpPanelPanel 类的 listFtpFiles()方法代码如下:

```
public synchronized void listFtpFiles(final TelnetInputStream list) {  
    final DefaultTableModel model = (DefaultTableModel) ftpDiskTable  
    .getModel();  
    model.setRowCount(0);  
    Runnable runnable = new Runnable() {  
        @Override  
        public synchronized void run() {  
            ftpDiskTable.clearSelection();  
            try {  
                String pwd = getPwd();  
                model.addRow(new Object[] { new FtpFile(".", pwd, true), "", "" });  
                // 添加 "." 符号  
                model.addRow(new Object[] { new FtpFile("..", pwd, true), "", "" });  
                // 添加 ".." 符号  
                BufferedReader br = new BufferedReader(new InputStreamReader  
                (list));  
                String data = null;  
                // 读取输入流中的文件目录  
                while ((data = br.readLine()) != null) {  
                    FtpFile ftpFile = new FtpFile();  
                    // 获取 FTP 服务器目录信息  
                    String dateStr = data.substring(0, 17).trim();  
                    String sizeOrDir = data.substring(18, 39).trim();  
                    String fileName = data.substring(39, data.length()).trim();  
                    // 将 FTP 目录信息初始化到 FTP 文件对象中  
                    ftpFile.setLastDate(dateStr);  
                    ftpFile.setSize(sizeOrDir);  
                    ftpFile.setName(fileName);  
                    ftpFile.setPath(pwd);  
                    model.addRow(new Object[] { ftpFile, ftpFile.getSize(),  
                    dateStr });  
                    // 将文件信息添加到表格中  
                }  
                br.close();  
            } catch (IOException ex) {  
                // 关闭输入流  
                Logger.getLogger(FTP_Client_Frame.class.getName()).log(Level.SEVERE, null, ex);  
            }  
        }  
    };  
    if (SwingUtilities.isEventDispatchThread())  
        runnable.run();  
    else  
        SwingUtilities.invokeLater(runnable);  
}
```

## 6.6 队列管理



### 6.6.1 功能概述

174



FTP 使用模块的队列是最核心的功能,它实现文件的上传和下载功能。除了以队列形式管理和排序文件传输任务外,队列管理还提供了文件上传、下载的记录,实现了任务队列的排序控制,以及任务的删除与清空、暂停等操作。队列管理使用一个选项卡面板来实





现, 任务队列管理界面如图 6.16 所示。

任务名称	方向	主机	执行状态	上移
ftp4_sap5.log	下载	192.168.1.109	正在下载	下移
/classpath	下载	192.168.1.109	等待下载	暂停
				删除
				清空

图 6.16 任务队列管理界面

## 6.6.2 任务队列

实现队列任务控制的步骤如下。

(1) 创建自定义的面板控件实现任务队列界面。本模块继承 JPanel 控件编写了新的面板控件 QueuePanel 类, 并且实现了 ActionListener 接口和该接口定义的事件处理方法。在该类的构造方法中创建了一个定时器控件, 该控件每间隔 1 秒检查一次任务队列的变化, 如果队列内容改变, 则调用 refreshQueue() 方法刷新队列。其关键代码如下:

```
public class QueuePanel extends JPanel implements ActionListener {
    private static final long serialVersionUID = 3767706397461683507L;
    private JTable queueTable = new JTable();           // 显示任务队列的表格组件
    private JScrollPane scrollPane = new JScrollPane();
    private FTP_Client_Frame frame;                     // 主窗体的引用对象
    private String[] columns;
    private FtpClient ftpClient;                       // FTP 协议的控制类
    private Timer queueTimer;                          // 队列的定时器
    private LinkedList<Object[]> localQueue;           // 本地面板的上传队列
    private LinkedList<Object[]> ftpQueue;             // FTP 面板的下载队列
    private JToggleButton stopButton;
    private boolean stop = false;                      // 队列的控制变量

    public QueuePanel() {
        initComponents();
    }

    public QueuePanel(FTP_Client_Frame frame) {
        this.frame = frame;
        // 从主窗体获取本地面板的上传队列
        localQueue = (LinkedList<Object[]>) frame.getLocalPanel().getQueue();
        // 从主窗体获取 FTP 面板的下载队列
        ftpQueue = (LinkedList<Object[]>) frame.getFtpPanel().getQueue();
        initComponents();                             // 初始化窗体界面
        // 创建定时器, 每间隔 1 秒执行队列刷新任务
        queueTimer = new Timer(1000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (localQueue.size() + ftpQueue.size() == queueTable
                    .getRowCount()) {
                    return;                             // 如果队列大小没有改变
                                                        // 结束本方法, 不做任何操作
                }
                refreshQueue();                         // 否则刷新显示队列信息的表格数据
            }
        });
    }
    // 省略部分代码
}
```

(2) 实现界面的按钮事件处理方法 actionPerformed(), 该方法由 ActionListener 接口定义, 由 QueuePanel 类实现其事件处理业务。它首先判断用户在界面中单击的是哪个按钮, 然后根据不同按钮, 执行不同的业务逻辑, 其中“上移”和“下移”按钮的事件处理







将调用其他方法完成。其关键代码如下：

```
public void actionPerformed(ActionEvent e) {
    final String command = e.getActionCommand();
    if (command.equals("stop&start")) {           // 处理暂停按钮事件
        if (stopButton.isSelected()) {
            stop = true;
            stopButton.setText("继续");
        } else {
            stop = false;
            stopButton.setText("暂停");
        }
    }
    // 处理上移和下移按钮事件
    if (command.equals("up") || command.equals("down")) {
        up_Down_Action(command);                  // 调用处理上移和下移动作的方法
    }
    if (command.equals("del")) {                   // 处理“删除”按钮的事件
        int row = queueTable.getSelectedRow();     // 获取显示队列的表格的当前选择行
        if (row < 0)
            return;
        // 获取选择行的第一个表格单元值
        Object valueAt = queueTable.getValueAt(row, 0);
        if (valueAt instanceof File) {             // 如果选择内容是 File 类的对象
            File file = (File) valueAt;
            int size = localQueue.size();           // 获取上传队列大小
            for (int i = 0; i < size; i++) {        // 遍历上传队列
                if (localQueue.get(i)[0].equals(file)) {
                    localQueue.remove(i);           // 从上传队列中删除文件对象
                }
            }
        } else if (valueAt instanceof String) {    // 如果选择的是字符串对象
            String fileStr = (String) valueAt;
            int size = ftpQueue.size();             // 获取上传队列的大小
            for (int i = 0; i < size; i++) {        // 遍历上传队列
                // 获取上传队列中的文件对象
                FtpFile ftpFile = (FtpFile) ftpQueue.get(i)[0];
                if (ftpFile.getAbsolutePath().equals(fileStr)) {
                    ftpQueue.remove(i);             // 从上传队列中删除该文件对象
                }
            }
        }
        refreshQueue();                            // 刷新队列列表
    }
    if (command.equals("clear")) {                  // 处理“清空”按钮的事件
        localQueue.clear();                        // 调用本地面板的队列的 clear() 方法
        ftpQueue.clear();                          // 调用 FTP 面板的队列的 clear() 方法
    }
}
```

(3) 编写“上移”和“下移”按钮的事件处理方法 `up_Down_Action()`，由于处理任务移动的业务代码比较复杂，并且代码较多，所以单独定义了一个方法来完成业务逻辑。其关键代码如下：

```
private void up_Down_Action(final String command) {
    int row = queueTable.getSelectedRow(); // 获取队列表格的当前选择行号
    if (row < 0)
        return;
    Object valueAt = queueTable.getValueAt(row, 0); // 获取当前选择行的第一个单元值
    // 获取当前选择行的第二个单元值作为判断上传或下载方向的依据
    String orientation = (String) queueTable.getValueAt(row, 1);
    if (orientation.equals("上传")) {           // 如果是上传任务
        String value = ((File) valueAt).getAbsolutePath();
        int size = localQueue.size();
        for (int i = 0; i < size; i++) {        // 遍历上传队列
            Object[] que = localQueue.get(i);
        }
    }
}
```







```

File file = (File) que[0];
// 从队列中遍历到选择的上传任务的文件对象
if (file.getAbsolutePath().equals(value)) {
    // 获取表格的选择模型
    ListSelectionModel selModel = queueTable.getSelectionModel();
    // 设置选择模型的单选模式
    selModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    int dsize = localQueue.size(); // 获取本地上传队列的大小
    int drow = queueTable.getSelectedRow(); // 获取队列表格的当前选择行号
    int queueVal = localQueue.size() - dsize;
    int next = -1;
    int selIndex = -1;
    if (command.equals("up")) {
        if (i < 1) // 限制选择范围
            return;
        selIndex = drow - queueVal - 1;
        next = i - 1;
    } else {
        if (i >= size - 1) // 限制选择范围
            return;
        selIndex = drow - queueVal + 1;
        next = i + 1;
    }
    // 交换两个队列位置的任务
    Object[] temp = localQueue.get(next);
    localQueue.set(next, que);
    localQueue.set(i, temp);
    refreshQueue(); // 刷新队列表格的列表
    selModel.setSelectionInterval(0, selIndex); // 设置表格选择第一行
    break;
}
} else if (orientation.equals("下载")) { // 如果是下载任务
    String value = (String) valueAt;
    int size = ftpQueue.size(); // 获取 FTP 下载队列的大小
    for (int i = 0; i < size; i++) { // 遍历下载队列
        Object[] que = ftpQueue.get(i);
        FtpFile file = (FtpFile) que[0]; // 获取每个下载任务的 FTP 文件对象
        if (file.getAbsolutePath().equals(value)) { //
            // 获取任务队列表格的选择模型
            ListSelectionModel selModel = queueTable.getSelectionModel();
            // 设置模型使用单选模式
            selModel.setSelectionMode(SINGLE_SELECTION);
            int dsize = ftpQueue.size();
            int drow = queueTable.getSelectedRow();
            int queueVal = ftpQueue.size() - dsize;
            int next = -1;
            int selIndex = -1;
            if (command.equals("up")) {
                if (i < 1) // 限制选择范围
                    return;
                selIndex = drow - queueVal - 1;
                next = i - 1;
            } else {
                if (i >= size - 1) // 限制选择范围
                    return;
                selIndex = drow - queueVal + 1;
                next = i + 1;
            }
            // 交换连个队列位置的任务内容
            Object[] temp = ftpQueue.get(next);
            ftpQueue.set(next, que);
            ftpQueue.set(i, temp);
            refreshQueue(); // 刷新任务队列的表格列表
            selModel.setSelectionInterval(0, selIndex); // 选择表格的第一行
            break;
        }
    }
}
}

```







```
}  
}  
}
```

(4) 编写刷新表格控件中队列信息的方法，该方法也被构造方法用于显示队列信息。它首先判断“自动关机”按钮是否被按下，如果按钮处于按下的状态，并且队列任务全部执行完毕，就执行关机命令。然后创建表格控件的数据模型，把上传队列和下载队列中的所有任务添加到数据模型中。最后把这个数据模型设置到表格控件的属性中。其关键代码如下：

```
private synchronized void refreshQueue() {  
    // 如果关机按钮被按下并且上传和下载的队列都没有任务  
    if (frame.getShutdownButton().isSelected() && localQueue.isEmpty() &&  
        ftpQueue.isEmpty()) {  
        try {  
            // 执行系统关机命令，延迟 30 秒钟  
            Runtime.getRuntime().exec("shutdown -s -t 30");  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
    // 创建表格的数据模型对象  
    DefaultTableModel model = new DefaultTableModel(columns, 0);  
    Object[] localQueueArray = localQueue.toArray(); // 获取本地上传队列中的任务  
    // 遍历本地上传任务  
    for (int i = 0; i < localQueueArray.length; i++) {  
        Object[] queueValue = (Object[]) localQueueArray[i];  
        if (queueValue == null)  
            continue;  
        File localFile = (File) queueValue[0];  
        // 把上传队列的任务添加到表格组件的数据模型中  
        model.addRow(new Object[] { localFile.getAbsolutePath(), "上传",  
            ftpClient.getServer(), i == 0 ? "正在上传" : "等待上传" });  
    }  
    Object[] ftpQueueArray = ftpQueue.toArray(); // 获取下载队列的任务  
    for (int i = 0; i < ftpQueueArray.length; i++) { // 遍历下载队列  
        Object[] queueValue = (Object[]) ftpQueueArray[i];  
        if (queueValue == null)  
            continue;  
        FtpFile ftpFile = (FtpFile) queueValue[0];  
        // 把下载队列的任务添加到表格组件的数据模型中  
        model.addRow(new Object[] { ftpFile.getAbsolutePath(), "下载",  
            ftpClient.getServer(), i == 0 ? "正在下载" : "等待下载" });  
    }  
    queueTable.setModel(model); // 设置表格使用本方法的表格数据模型  
}
```

### 6.6.3 本地队列文件上传

实现队列文件上传的程序界面很简单，但是该界面的内容由上传线程控制，这个上传线程才是核心内容。它负责读取本地资源管理面板中的任务队列（也就是上传队列）中的所有上传文件的任务，然后按照队列的顺序去执行它们，同时更新每个任务在界面中的状态，如进度条的进度、完成情况等。其关键代码如下：

```
class UploadThread extends Thread {
```



```

private LocalPanel localPanel;
String path = ""; // 上传文件的本地相对路径
String selPath; // 选择的本地文件的路径
private boolean conRun = true; // 线程的控制变量
private FtpClient ftpClient; // FTP 控制类
private Object[] queueValues; // 队列任务数组
public UploadThread(LocalPanel localPanel, String server, int port,
String userStr, String passStr) {
    try {
        ftpClient = new FtpClient(server, port);
        ftpClient.login(userStr, passStr);
        ftpClient.binary();
        path = ftpClient.pwd();
    } catch (IOException e) {
        e.printStackTrace();
    }
    this.localPanel = localPanel;
    new Thread() { // 创建保持服务器通信的线程
        @Override
        public void run() {
            while (conRun) {
                try {
                    Thread.sleep(30000);
                    // 定时向服务器发送消息, 保持连接
                    UploadThread.this.ftpClient.noop();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }.start();

    public void stopThread() { // 停止线程的方法
        conRun = false;
    }

    private void copyFile(File file, FtpFile ftpFile) { // 递归遍历文件夹的方法
        // 判断队列面板是否执行暂停命令
        while (localPanel.frame.getQueuePanel().isStop()) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        Object[] args = localPanel.queue.peek();
        // 判断队列顶是不是上一个处理的任务
        if (queueValues == null || args == null || !queueValues[0]
.equals(args[0]))
            return;
        try {
            path = file.getParentFile().getPath().replace(selPath, "");
            ftpFile.setName(path.replace("\\", "/"));
            path = ftpFile.getAbsolutePath();
            System.out.println(path);
            if (file.isFile()) {
                UploadPanel uploadPanel = localPanel.frame.getUploadPanel();
                // 远程 FTP 的文件名绝对路径
                String remoteFile = path + "/" + file.getName();
                System.out.println("remoteFile:" + remoteFile);
                double fileLength = file.length() / Math.pow(1024, 2);
                ProgressArg progressArg = new ProgressArg((int) (file.length()
/ 1024), 0, 0);
                String size = String.format("%.4f MB", fileLength);
                Object[] row = new Object[] { file.getAbsolutePath(), size,
remoteFile,
                ftpClient.getServer(), progressArg };
                uploadPanel.addRow(row);
            }
        }
    }
}

```





```
// 获取服务器文件的输出流
OutputStream put = ftpClient.put(remoteFile);
FileInputStream fis = null; // 本地文件的输入流
try {
    fis = new FileInputStream(file); // 初始化文件的输入流
} catch (Exception e) {
    e.printStackTrace();
    return;
}
int readNum = 0;
byte[] data = new byte[1024]; // 缓存大小
while ((readNum = fis.read(data)) > 0) { // 读取本地文件到缓存
    Thread.sleep(0, 30); // 线程休眠
    put.write(data, 0, readNum); // 输出到服务器
    progressArg.setValue(progressArg.getValue() + 1); // 累加进度条
}
progressArg.setValue(progressArg.getMax()); // 结束进度条
fis.close(); // 关闭文件输入流
put.close(); // 关闭服务器输出流
} else if (file.isDirectory()) {
    path = file.getPath().replace(selPath, "");
    ftpFile.setName(path.replace("\\", "/"));
    ftpClient.sendServer("MKD." + path + "\r\n");
    ftpClient.readServerResponse();
    File[] listFiles = file.listFiles();
    for (File subFile : listFiles) {
        Thread.sleep(0, 50);
        copyFile(subFile, ftpFile);
    }
}
} catch (FileNotFoundException e1) {
    e1.printStackTrace();
    System.exit(0);
} catch (Exception ex) {
    ex.printStackTrace();
}
}
@Override
public void run() { // 线程的主体方法
    while (conRun) {
        try {
            Thread.sleep(1000); // 线程休眠 1 秒
            Queue<Object> queue = localPanel.queue; // 获取本地面板的队列对象
            queueValues = queue.peek(); // 获取队列首的对象
            if (queueValues == null) { // 如果该对象为空
                continue; // 进行下一次循环
            }
            File file = (File) queueValues[0]; // 获取队列中的本队文件对象
            // 获取队列中的 FTP 文件对象
            FtpFile ftpFile = (FtpFile) queueValues[1];
            if (file != null) {
                selPath = file.getParent();
                copyFile(file, ftpFile); // 调用递归方法上传文件
                FtpPanel ftpPanel = localPanel.frame.getFtpPanel();
                ftpPanel.refreshCurrentFolder(); // 刷新 FTP 面板中的资源
            }
            Object[] args = queue.peek();
            // 判断队列项是否为处理的上一个任务。
            if (queueValues == null || args == null || !queueValues[0]
                .equals(args[0])) {
                continue;
            }
            queue.remove(); // 移除队列首元素
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```







```
    }
}
```

队列管理中上传面板由 UploadPanel 类实现，它继承了 JPanel 控件，并用表格控件显示上传文件信息。该类提供的唯一方法用于向表格控件增加数据。该方法的关键代码如下：

```
public void addRow(final Object[] values) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            model.insertRow(0, values);          // 向表格的数据模型添加数据
        }
    };
    if (SwingUtilities.isEventDispatchThread())
        runnable.run();                        // 在事件队列执行
    else
        SwingUtilities.invokeLater(runnable);   // 或有事件队列调用
}
```

#### 6.6.4 FTP 队列文件下载

与队列文件上传的功能类似，实现队列文件下载的程序界面也很简单，但是该界面的内容由下载线程控制。它负责读取 FTP 资源管理面板中的任务队列（也就是下载队列）中的所有下载文件的任务，然后按照队列的顺序去执行它们，同时更新每个任务在界面中的状态。其关键代码如下：

```
public class DownThread extends Thread {
    private final FtpPanel ftpPanel;          // FTP 资源管理面板
    private final FtpClient ftpClient;        // FTP 控制类
    private boolean conRun = true;            // 线程的控制变量
    private String path;                      // FTP 的路径信息
    private Object[] queueValues;             // 下载任务的数组
    public DownThread(FtpPanel ftpPanel) {
        this.ftpPanel = ftpPanel;
        ftpClient = new FtpClient();          // 创建新的 FTP 控制对象
        FtpClient ftp = ftpPanel.ftpClient;
        try {
            // 连接到 FTP 服务器
            ftpClient.openServer(ftp.getServer(), ftp.getPort());
            ftpClient.login(ftp.getName(), ftp.getPass()); // 登录服务器
            ftpClient.binary();                 // 使用二进制传输
            ftpClient.noop();
        } catch (IOException e) {
            e.printStackTrace();
        }
        new Thread() {                        // 创建保持服务器通信的线程
            @Override
            public void run() {
                while (conRun) {
                    try {
                        Thread.sleep(30000);
                        ftpClient.noop();        // 定时向服务器发送消息，保持连接
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();
    }
    public void stopThread() {                // 停止线程的方法
        conRun = false;
    }
}
```







```
}
private void downFile(FtpFile file, File localFolder) {
    // 判断队列面板是否执行暂停命令
    while (ftpPanel.frame.getQueuePanel().isStop()) {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    Object[] args = ftpPanel.queue.peek();
    // 判断队列项是否为处理的上一个任务
    if (queueValues == null || args == null || !queueValues[0]
        .equals(args[0]))
        return;
    try {
        String ftpFileStr = file.getAbsolutePath().replaceFirst
            (path + "/", "");
        if (file.isFile()) {
            // 获取服务器指定文件的输入流
            TelnetInputStream ftpIs = ftpClient.get(file.getName());
            if (ftpIs == null) {
                JOptionPane.showMessageDialog(this.ftpPanel,
                    file.getName() + "无法下载");
                return;
            }
            // 创建本地文件对象
            File downFile = new File(localFolder, ftpFileStr);
            // 创建本地文件的输出流
            FileOutputStream fout = new FileOutputStream(downFile, true);
            // 计算文件大小
            double fileLength = file.getLongSize() / Math.pow(1024, 2);
            ProgressArg progressArg = new ProgressArg((int) (file.getLongSize()
                / 1024), 0, 0);
            String size = String.format("%.4f MB", fileLength);
            Object[] row = new Object[] { ftpFileStr, size,
                downFile.getAbsolutePath(),
                    ftpClient.getServer(), progressArg };
            DownloadPanel downloadPanel = ftpPanel.frame.getDownloadPanel();
            downloadPanel.addRow(row);
            byte[] data = new byte[1024]; // 定义缓存
            int read = -1;
            while ((read = ftpIs.read(data)) > 0) { // 读取FTP文件内容到缓存
                Thread.sleep(0, 30); // 线程休眠
                fout.write(data, 0, read); // 将缓存数据写入本地文件
                // 累加进度条
                progressArg.setValue(progressArg.getValue() + 1);
            }
            progressArg.setValue(progressArg.getMax()); // 结束进度条
            fout.close(); // 关闭文件输出流
            ftpIs.close(); // 关闭FTP文件输入流
        } else if (file.isDirectory()) { // 如果下载的是文件夹
            File directory = new File(localFolder, ftpFileStr);
            // 创建本地文件夹对象
            directory.mkdirs(); // 创建本地的文件夹
            ftpClient.cd(file.getName()); // 改变FTP服务器的当前路径
            // 获取FTP服务器的文件列表信息
            InputStreamReader list = new InputStreamReader(ftpClient.list());
            BufferedReader br = new BufferedReader(list);
            String nameStr = null;
            while ((nameStr = br.readLine()) != null) {
                Thread.sleep(0, 50);
                String name = nameStr.substring(39);
                String size = nameStr.substring(18, 39);
                FtpFile ftpFile = new FtpFile(); // 创建FTP文件对象
                ftpFile.setName(name); // 设置文件名
                ftpFile.setPath(file.getAbsolutePath()); // 设置文件路径
            }
        }
    }
}
```







```

        ftpFile.setSize(size);           // 设置文件大小
        downFile(ftpFile, localFolder);  // 递归执行子文件夹的下载
    }
    list.close();
    br.close();
    ftpClient.cdUp();                    // 返回 FTP 上级路径
}
} catch (Exception ex) {
    ex.printStackTrace();
}
}
@Override
public void run() {                      // 线程业务方法
    while (conRun) {
        try {
            Thread.sleep(1000);
            ftpClient.noop();
            queueValues = ftpPanel.queue.peek();
            if (queueValues == null) {
                continue;
            }
            FtpFile file = (FtpFile) queueValues[0];
            File localFolder = (File) queueValues[1];
            if (file != null) {
                path = file.getPath();
                ftpClient.cd(path);
                downFile(file, localFolder);
                path = null;
                ftpPanel.frame.getLocalPanel().refreshCurrentFolder();
            }
            Object[] args = ftpPanel.queue.peek();
            // 判断队列项是否为处理的上一个任务。
            if (queueValues == null || args == null || !queueValues[0]
                .equals(args[0]))
                continue;
            ftpPanel.queue.poll();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    System.out.println("thread is UnAlive!");
}
}

```

队列管理中下载面板由 DownloadPanel 类实现, 它简单的继承 JPanel 控件实现自定义的面板, 并在面板上布置显示已经下载 FTP 文件信息的表格控件。该类提供的唯一方法, 就是向表格控件添加一行数据的 addRow() 方法。它将被上传线程调用, 其关键代码如下:

```

public void addRow(final Object[] values) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            model.insertRow(0, values);
        }
    };
    if (SwingUtilities.isEventDispatchThread())
        runnable.run();
    else
        SwingUtilities.invokeLater(runnable);
}

```



# 第 7 章

---

## 局域网通信模块

( Swing+Java DB 实现 )

近年来，各种局域网通信软件得到了飞速发展，它可以不用连接 Internet，直接在局域网内实现信息通信、工作交流、提交计划等业务。这种通信系统广泛应用于中、小型企业的内部通信，可以大大提高员工的工作效率，在方便企业内部员工交流的同时，也创造了一个安静的工作环境，是现代企业不可缺少的辅助工具。

本章将介绍如何使用 Java Swing 技术和 Java 6.0 新增的 Java DB 数据库开发跨平台的应用程序。通过阅读本章，读者可以学习到：

- » 使用 UDP 通信协议
- » JavaDB 数据库的应用
- » 如何使用 Java 6.0 的系统托盘
- » 如何实现多点通信





## 7.1 局域网通信概述

### 7.1.1 模块概述

局域网通信是一种采用 UDP 通信协议实现在局域网内部进行通信的模块, UDP 是无连接的通信协议, 传输效率高, 但是不能保证信息一定会到达目的地, 因此是一种不可靠的通信协议。本章针对局域网通信的基本操作制作了一个局域网通信程序。该程序可以在局域网内实现信息通信、工作交流、提交计划等业务。这种通信系统广泛应用于中、小型企业的内部通信, 可以大大提高员工的工作效率, 在方便企业内部员工交流的同时, 也创造了一个安静的工作环境, 是现代企业不可缺少的辅助工具。

### 7.1.2 功能结构

局域网通信模块可以实现信息通信、系统升级、系统设置、用户搜索、访问公共资源和访问对方主机等功能, 其功能结构如图 7.1 所示。

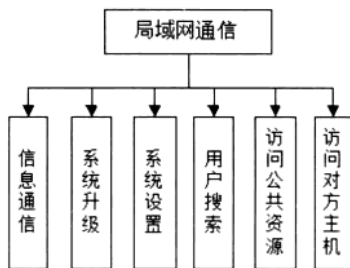


图 7.1 局域网通信模块的功能结构

### 7.1.3 程序预览

局域网通信模块由多个程序界面组成, 下面仅列出几个典型界面的预览, 其他界面参见光盘中的源程序。局域网通信模块的主界面如图 7.2 所示, 该界面包含调用所有功能模块的控件。通信窗体的界面如图 7.3 所示, 该界面用于发送和接收通信信息; 另外, 还可以在对方未开启局域网通信程序的情况下, 向对方发送信使信息。



图 7.2 主窗体

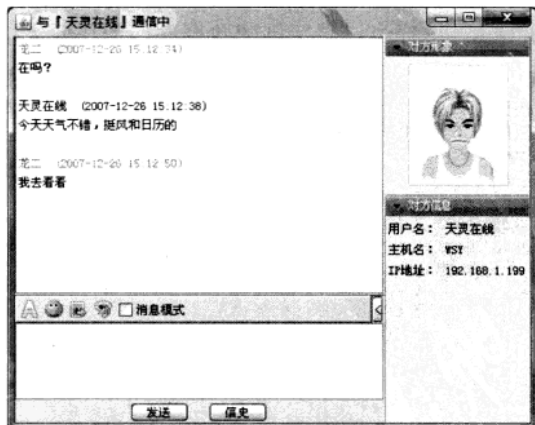


图 7.3 通信窗体界面





模块的工具界面如图 7.4 所示, 该界面主要用于更换界面外观、搜索用户和软件更新。模块设置界面如图 7.5 所示, 该界面用于设置系统路径、登录模块、IP 搜索范围等。

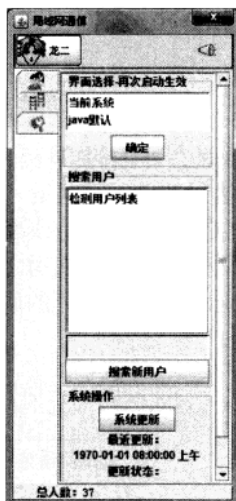


图 7.4 模块的工具界面

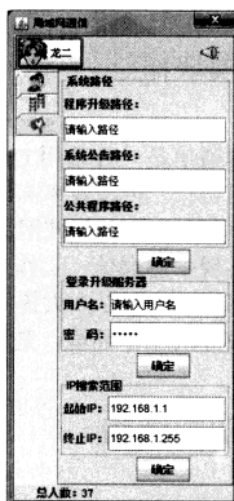


图 7.5 模块设置界面

## 7.2 关键技术

### 7.2.1 创建操作数据库的 Dao 类

Dao 类主要负责有关数据库的操作, 该类在构造方法中加载驱动并连接数据库, 然后将构造方法设置为 private 私有属性, 再创建一个静态的 getDao() 方法用来获取 Dao 类的实例对象, 这是典型的单例模式。

连接数据库时, 可以在连接数据库的 URL 后面添加 “create=true”, 将 create 参数设置为 true, 这样可以直接创建数据库, 但在此之前需要调用 dbExists() 方法判断数据库是否存在。Dao 类的关键代码如下:

```
public class Dao {  
    //数据库驱动  
    private static final String driver = "org.apache.derby.jdbc.EmbeddedDriver";  
    private static String url = "jdbc:derby:db_EQ"; //数据库 URL  
    private static Connection conn = null; //数据库连接  
    private static Dao dao = null;  
    private Dao() {  
        try {  
            Class.forName(driver); //加载驱动  
            if (!dbExists()) { //如果数据库不存在  
                //创建数据库  
                conn = DriverManager.getConnection(url + ";create=true");  
                createTable(); //创建数据表  
            } else  
                conn = DriverManager.getConnection(url); //连接数据库  
            addDefUser(); //添加本地用户到数据库  
        } catch (Exception e) {  
            //处理异常  
        }  
    }  
}
```







```

        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "数据库连接异常, 或者本软件已经运行。");
        System.exit(0);
    }
}

private boolean dbExists() { //检测数据库的方法
    boolean bExists = false;
    File dbFileDir = new File("db_EQ");
    if (dbFileDir.exists()) {
        bExists = true;
    }
    return bExists;
}

public static Dao getDao() { //获取 Dao 实例的方法
    if (dao == null)
        dao = new Dao();
    return dao;
}

public void createTable() { //创建数据表的方法
    String createUserSql = "CREATE TABLE tb_users ("
        + "ip varchar(16) primary key," + "host varchar(30),"
        + "name varchar(20)," + "tooltip varchar(50),"
        + "icon varchar(50))";
    String createLocationSql = "CREATE TABLE tb_location ("
        + "xLocation int," + "yLocation int," + "width int,"
        + "height int)";

    try {
        Statement stmt = conn.createStatement();
        stmt.execute(createUserSql); //创建用户数据表
        stmt.execute(createLocationSql); //创建窗体定位数据表
        addDefLocation();
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

### 1. addDefLocation()方法

该方法把窗体的默认位置和大小添加到数据库保存起来,在局域网通信模块首次运行时,将使用该方法设置窗体位置和窗体大小。如果用户更改了窗体位置或者窗体大小,该方法所保存的数据将不再起作用。关键代码如下:

```

public void addDefLocation() { //添加默认窗体位置
    String sql = "insert into tb_location values(?,?,?,?)"; //定义带参数的 SQL 语句
    try {
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setInt(1, 100); //传递第一个 SQL 参数
        pst.setInt(2, 0); //传递第二个 SQL 参数
        pst.setInt(3, 240); //传递第三个 SQL 参数
        pst.setInt(4, 500); //传递第四个 SQL 参数
        pst.executeUpdate(); //执行 SQL 更新
        pst.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

### 2. addDefUser()方法

该方法使用本机 IP 地址创建默认用户,并添加到数据库中。除了使用本机 IP 地址之外,默认的用户还包括主机名称、姓名、提示文本和头像图标等属性。关键代码如下:

```

public void addDefUser() { //创建本机用户
    try {

```







```
InetAddress local = InetAddress.getLocalHost(); //获取本机地址对象
User user = new User();                       //创建新用户对象
user.setIp(local.getHostAddress());           //初始化用户对象
user.setHost(local.getHostName());
user.setName(local.getHostName());
user.setTipText(local.getHostAddress());
user.setIcon("1.gif");
if (getUser(user.getIp()) == null) {
    addUser(user);                             //添加该用户到数据库中
}
} catch (UnknownHostException e) {
    e.printStackTrace();
}
}
```

### 3. getLocation()方法

该方法用于从数据库中获取窗体位置和窗体大小信息，并将这些信息封装成 Rectangle 类的实例对象，然后作为方法的返回值。关键代码如下：

```
public Rectangle getLocation() { //获取窗体位置
    Rectangle rec = new Rectangle(100, 0, 240, 500); //创建 rec 对象并带有默认数据
    String sql = "select * from tb_location";
    try {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql); //从数据库读取数据
        if (rs.next()) {
            rec.x = rs.getInt(1); //初始化 rec 对象
            rec.y = rs.getInt(2);
            rec.width = rs.getInt(3);
            rec.height = rs.getInt(4);
        }
        rs.close();
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rec;
}
```

### 4. addUser()方法

该方法用于添加指定用户到数据库中，方法接收 User 类的实例对象，即用户对象作为参数，在 SQL 语句中将用户对象的属性作为参数插入到数据库中。关键代码如下：

```
public void addUser(User user) { //添加用户
    try {
        String sql = "insert into tb_users values(?,?,?,?)"; //添加用户的 SQL 语句
        PreparedStatement ps = null;
        ps = conn.prepareStatement(sql);
        ps.setString(1, user.getIp()); //填充 SQL 语句的参数
        ps.setString(2, user.getHost());
        ps.setString(3, user.getName());
        ps.setString(4, user.getTipText());
        ps.setString(5, user.getIcon());
        ps.execute(); //执行 SQL 语句
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### 5. delUser()方法

该方法用于从数据库中删除指定用户的记录，方法接收 User 类的实例对象参数，并以该用户对象的 IP 属性为查询条件，从数据库中删除指定 IP 的用户信息。关键代码如下：







```

public void delUser(User user) { //删除用户
    try {
        String sql = "delete from tb_users where ip=?"; //删除用户的 SQL 语句
        PreparedStatement ps = null;
        ps = conn.prepareStatement(sql);
        ps.setString(1, user.getIp()); //填充 SQL 语句的参数
        ps.executeUpdate(); //执行 SQL 语句
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

### 6. getUser()方法

该方法用于从数据库获取指定 IP 的用户对象,该用户对象包含了指定 IP 地址的用户的所有属性,并且将这些属性封装到用户对象中,然后作为方法的返回值。关键代码如下:

```

public User getUser(String ip) { //获取指定 IP 的用户
    String sql = "select * from tb_users where ip=?"; //定义查询用户表的 SQL 语句
    User user = null;
    try {
        PreparedStatement pst = conn.prepareStatement(sql);
        pst.setString(1, ip);
        ResultSet rs = pst.executeQuery(); //执行用户查询
        if (rs.next()) {
            user = new User(); //创建用户对象
            user.setIp(rs.getString(1)); //初始化用户对象
            user.setHost(rs.getString(2));
            user.setName(rs.getString(3));
            user.setTipText(rs.getString(4));
            user.setIcon(rs.getString(5));
        }
        rs.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return user; //返回用户对象
}

```

### 7. updateLocation()方法

该方法用于更新窗体位置和窗体大小信息,方法将接收 Rectangle 类的实例对象作为参数,将位置、宽度和高度等参数保存到数据库中。关键代码如下:

```

public void updateLocation(Rectangle location) { //更新窗体位置
    String sql = "update tb_location set xLocation=?,yLocation=?,"
        + "width=?,height=?";
    try {
        PreparedStatement pst
            = conn.prepareStatement(sql); //初始化 SQL 语句的参数
        pst.setInt(1, location.x);
        pst.setInt(2, location.y);
        pst.setInt(3, location.width);
        pst.setInt(4, location.height);
        pst.executeUpdate(); //执行 SQL 更新语句
        pst.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

### 8. updateUser()方法

该方法用于更新用户信息,其中可更新的内容包括除 IP 地址以外的所有信息,如用户的主机名称、头像、姓名等。关键代码如下:







```
public void updateUser(User user) { //修改用户
    try {
        String sql = "update tb_users set host=?,name=?,tooltip=?,icon=?
        where ip='" + user.getHost() + "'";
        PreparedStatement ps = null;
        ps = conn.prepareStatement(sql);
        ps.setString(1, user.getHost()); //初始化 SQL 语句的参数
        ps.setString(2, user.getName());
        ps.setString(3, user.getTipText());
        ps.setString(4, user.getIcon());
        ps.executeUpdate(); //执行 SQL 更新语句
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

## 7.2.2 工具类的实现

Resource 类是局域网通信模块中的工具类，该类中的工具方法都是静态的，可以直接调用，而不用创建 Resource 类的实例对象。这些工具方法包括搜索用户的方法、登录公共资源的方法、信使群发的方法和单条信息发送的方法。

### 1. searchUsers()方法

该方法用于搜索局域网中的通信用户，也就是搜索企业中的所有员工。该方法将获取用户指定的 IP 搜索范围，并在该范围内搜索所有可以访问的计算机，如果用户没有指定 IP 范围，那么 IP 地址默认的范围是 192.168.0.1~192.168.1.255。searchUsers()方法的关键代码如下：

```
public static void searchUsers(ChatTree tree, JProgressBar progressBar,
    JList list, JToggleButton button) {
    String ipStart = EQ.preferences.get("ipStart", "192.168.0.1");
    String ipEnd = EQ.preferences.get("ipEnd", "192.168.1.255");
    String[] is = ipStart.split("\\.");
    String[] ie = ipEnd.split("\\.");
    int[] ipsInt = new int[4];
    int[] ipeInt = new int[4];
    for (int i = 0; i < 4; i++) {
        ipsInt[i] = Integer.parseInt(is[i]);
        ipeInt[i] = Integer.parseInt(ie[i]);
    }
    progressBar.setIndeterminate(true);
    progressBar.setStringPainted(true);
    DefaultListModel model = new DefaultListModel();
    model.addElement("搜索结果: ");
    list.setModel(model);
    try {
        for (int l = ipsInt[0]; l <= ipeInt[0]; l++) {
            boolean b0 = l < ipeInt[0]; //记录第一层循环的条件
            int k = l != ipsInt[0] ? 0 : ipsInt[1]; //从第二次循环以后 k 赋值 0
            for (; b0 ? k < 256 : k <= ipeInt[1]; k++) {
                boolean b1 = b0 || k < ipeInt[1]; //记录第二层循环的条件
                int j = k != ipsInt[1] ? 0 : ipsInt[2]; //从第二次循环以后 j 赋值 0
                for (; b1 ? j < 256 : j <= ipeInt[2]; j++) {
                    boolean b2 = b1 || b1 ? j < 256 : j < ipeInt[2];
                    int i = j != ipsInt[2] ? 0 : ipsInt[3];
                    for (; b2 ? i < 256 : i <= ipeInt[3]; i++) {
                        if (!button.isSelected()) { //如果“搜索新用户”按钮没有被选择
```







```

        progressBar.setIndeterminate(false); //取消进度条的滚动
        return;
    }
    Thread.sleep(100); //线程休息 100 毫秒
    String ip = l + "." + k + "." + j + "." + i;
    progressBar.setString("正在搜索: " + ip); //设置进度条文本
    if (tree.addUser(ip, "search"))
        model.addElement("<html><b><font color=green>添加"
            //添加新用户
            + ip + "</font></b></html>");
    }
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    progressBar.setIndeterminate(false); //停止进度条
    progressBar.setString("搜索完毕"); //进度条显示搜索完毕
    button.setText("搜索新用户"); //恢复“搜索新用户”按钮文本
    button.setSelected(false); //恢复“搜索新用户”按钮状态
}
}

```

## 2. loginPublic()方法

该方法用于登录程序升级的服务器，它获取用户指定的升级路径、用户名和密码，使用“net use”命令访问服务器，并返回访问成功或者失败的 boolean 值。关键代码如下：

```

public static boolean loginPublic(String user, String pass) {
    try {
        String userName = user;
        String updatePath = EQ.preferences.get("updatePath", null);
        //获取升级路径
        if (updatePath == null) //如果未指定路径
            return false; //返回失败结果
        File file = new File(updatePath);
        if (!file.exists())
            return false;
        if (file.isFile())
            updatePath = file.getParent();
        if (userName != null && !userName.equals(""))
            userName = "/" + user + userName;
        Process process = Runtime.getRuntime().exec(
            "cmd /c %windir% + File.separator + "System32"
            + File.separator + "net use " + updatePath + " "
            + pass + userName); //访问升级服务器
        Scanner sce = new Scanner(process.getErrorStream());
        StringBuilder stre = new StringBuilder();
        while (sce.hasNextLine()) {
            stre.append(sce.nextLine()); //获取访问结果
        }
        process.destroy();
        String resulte = stre.toString();
        if (resulte.equals("")) //如果没有访问结果
            return true; //访问成功
        else //否则
            JOptionPane.showMessageDialog(EQ.frame, resulte, "错误信息",
                JOptionPane.ERROR_MESSAGE); //提示错误信息
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}

```

## 3. sendMessenger()方法

该方法用于发送信使到指定的用户，当通信对方没有运行局域网通信模块时，就无法







接收到通信模块发送的内容，这时可以调用该方法向对方发送信使，对方的操作模块接收到信使之后会以对话框的通知方式显示信使内容。关键代码如下：

```
//发送信使信息
public static void sendMessage(User user, String message, TelFrame frame)
{
    class TheThread implements Runnable { //定义内部线程类
    ...//省略部分代码
        public void run() {
            try {
                sendButton.setEnabled(false); //取消“发送”按钮的状态
                Process process = Runtime.getRuntime().exec(
                    "net send " + user.getIp() + " " + message);
                //执行信使发送命令
                InputStream is = process.getInputStream();
                int i, j;
                StringBuilder sb = new StringBuilder();
                while ((i = is.read()) != -1) {
                    sb.append((char) i); //获取信使发送结果
                }
                String runIs = new String(sb.toString().getBytes(
                    "iso-8859-1")).trim().replace(user.getIp(),
                    user.getName()); //将结果信息转码
                InputStream eis = process.getErrorStream();
                StringBuilder esb = new StringBuilder();
                while ((j = eis.read()) != -1) {
                    esb.append((char) j); //获取信使发送的错误
                }
                String runEis = new String(esb.toString().getBytes(
                    "iso-8859-1")).trim().replace(user.getIp(),
                    user.getName()); //将错误信息转码
                //显示信使发送结果
                frame.appendReceiveText(runIs, new Color(187, 30, 193));
                if (runEis.length() > 0) //如果存在错误信息
                    frame.appendReceiveText(runIs, Color.RED); //提示发送失败
                sendButton.setEnabled(true); //恢复“发送”按钮的状态
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    new Thread(new TheThread(user, message, frame)).start();
}
```

#### 4. sendGroupMessenger()方法

该方法可以向指定用户群体发送信使，如果企业中有针对会议参与人员的信息或者个别群体员工的信息，可以使用该方法向指定群体发送信使。群发信使的界面如图 7.6 所示。

sendGroupMessenger()方法的关键代码如下：

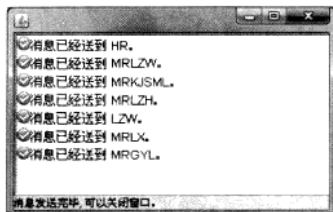


图 7.6 群发信使界面

```
public static void sendGroupMessenger(final TreePath[] selectionPaths, final
String message) { //群发信使信息
    new Thread(new Runnable() { //创建内部线程
        int bufferSize = 512;
```







```

    public void run() {
        MessageFrame messageFrame = new MessageFrame(); //创建信使窗口
        try {
            for (TreePath path : selectionPaths) { //遍历选择的用户
                DefaultMutableTreeNode node = (DefaultMutableTreeNode) path.
getLastPathComponent();
                User user = (User) node.getUserObject();
                messageFrame.setStateBarInfo("<html>正在给<font color=blue>"
+ user.getName()+ "</font> 发送消息 ... ..
</html>"); //提示发送信使
                Thread.sleep(20); //线程休眠
                InetAddress addr
                    = InetAddress.getByName(user.getIp());
                if (!addr.getHostAddress().equals(addr.getHostName())) {
                    Process process
                        = Runtime.getRuntime()
                            .exec(
                                //发送信使
                                "net send " + user.getIp() + " " + message);
                    InputStream is = process.getInputStream();
                    int i;
                    String sb = null;
                    byte[] data = new byte[bufferSize];
                    if ((i = is.read(data)) != -1) {
                        sb = new String(data, 0, i); //获取发送结果
                    }
                    String runIs = sb;
                    runIs = runIs.replace(user.getIp(), user.getName()).trim();
                    process.destroy();
                    if (runIs.indexOf("出错") < 0) //如果结果包含错误
                        messageFrame.addMessage(runIs, true); //显示错误信息
                    else //否则
                        messageFrame.addMessage(runIs, false); //显示成功信息
                } else {
                    messageFrame.addMessage("错误: " + user.getName()
+ "可能没有开机或启动了防火墙", false); //发送失败的提示
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //提示信使群发完毕
        messageFrame.setStateBarInfo("消息发送完毕,可以关闭窗口。");
    }
}).start();
}

```

### 5. startFolder()方法

该方法用于打开指定的文件夹或者网络共享资源。它通过“cmd /c start”指令打开 str 参数指定的文件夹位置。关键代码如下:

```

public static void startFolder(String str) {
    try {
        Runtime.getRuntime().exec("cmd /c start " + str);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```





## 7.3 主窗体

### 7.3.1 功能概述

主窗体界面也是局域网通信模块的用户列表，它由用户列表、公告提示、选项卡等组成。其中选项卡用于切换不同管理界面，包括系统操作、系统设置和用户列表三个界面。主窗体的运行效果如图 7.7 所示。

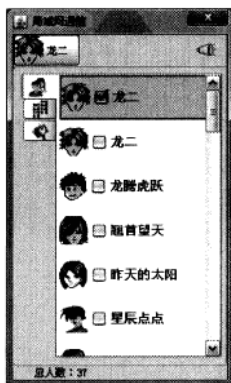


图 7.7 主窗体界面

### 7.3.2 实现主窗体

实现主窗体的步骤如下：

(1) 创建名为 EQ 的窗体类，为窗体添加选项卡面板，并添加用户列表、系统工具、系统设置 3 个选项卡和状态栏标签、公告按钮等属性。关键代码如下：

```
public class EQ extends Dialog {
    private JTextField ipEndTField;           //起始 ip 文本框
    private JTextField ipStartTField;         //终止 ip 文本框
    private JTextField userNameTField;        //用户名文本框
    private JPasswordField passwordTField;    //密码文本框
    private JTextField placardPathTField;     //公告路径文本框
    private JTextField updatePathTField;      //更新路径文本框
    private JTextField pubPathTField;         //公共程序路径文本框
    public static EQ frame = null;           //主窗体对象
    private ChatTree chatTree;               //用户树对象
    private JPopupMenu popupMenu;            //弹出菜单对象
    private JTabbedPane tabbedPane;          //选项卡
    private JToggleButton searchUserButton;  //搜索用户按钮
    private JProgressBar progressBar;        //搜索进度条
    private JList faceList;                  //搜索列表
    private JButton selectInterfaceOKButton; //界面选择按钮
    private DatagramSocket ss;               //数据通信包
    private final JLabel stateLabel;          //状态栏标签
    private static String user_dir;          //用户当前文件夹
    private static File localFile;           //本地文件
    private static File netFile;             //升级路径上的网络文件
    private String netFilePath;              //升级文件路径
    private JButton messageAlertButton;      //公告信息按钮
    private Stack<String> messageStack;      //公告信息栈
    private ImageIcon messageAlertIcon;      //公告信息图标
```







```
private ImageIcon messageAlertNullIcon;           //公告信息空图标
private Rectangle location;                       //窗体位置
public static TrayIcon trayicon;                 //系统托盘
private Dao dao;                                 //数据库操作类
//首选项
public final static Preferences preferences = Preferences.systemRoot();
private JButton userInfoButton;                  //本地用户按钮
...//省略部分代码
}
```

(2) 在构造方法中初始化窗体上的控件、数据库操作类、首选项对象等属性, 另外还要为窗体添加事件监听器、为公告信息按钮添加事件监听器等。关键代码如下:

```
public EQ() {
    super(new Frame());
    frame = this;                                //初始化窗体对象
    dao = Dao.getDao();                          //初始化 Dao 对象
    location = dao.getLocation();                 //初始化窗体位置对象
    setTitle("局域网通信");                      //设置窗体标题
    setBounds(location);                         //设置窗体位置
    progressBar = new JProgressBar();            //初始化搜索用户进度条
    progressBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
    tabbedPane = new JTabbedPane();              //初始化选项卡
    popupMenu = new JPopupMenu();                //初始化弹出菜单
    chatTree = new ChatTree(this);
    user_dir = System.getProperty("user.dir");    //用户当前路径
    localFile = new File(user_dir + File.separator + "EQ.jar"); //本地 EQ 文件
    stateLabel = new JLabel();                   //状态栏标签
    addWindowListener(new FrameWindowListener()); //添加窗体监视器
    {                                             //初始化公告信息按钮
        messageAlertIcon = new ImageIcon(EQ.class.getResource
            ("/image/messageAlert.gif"));
        messageAlertNullIcon = new ImageIcon(EQ.class.getResource
            ("/image/messageAlertNull20.gif"));
        messageStack = new Stack<String>();
        messageAlertButton = new JButton();
        messageAlertButton.setHorizontalAlignment(SwingConstants.RIGHT);
        messageAlertButton.setContentAreaFilled(false);
        final JPanel BannerPanel = new JPanel();
        BannerPanel.setLayout(new BorderLayout());
        add(BannerPanel, BorderLayout.NORTH);
        userInfoButton = new JButton();
        BannerPanel.add(userInfoButton, BorderLayout.WEST);
        userInfoButton.setMargin(new Insets(0, 0, 0, 10));
        initUserInfoButton();                    //初始化本地用户头像按钮
        BannerPanel.add(messageAlertButton, BorderLayout.CENTER);
        //添加公告信息按钮监听器
        messageAlertButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                if (!messageStack.empty()) {
                    showMessageDialog(messageStack.pop()); //显示公告信息
                }
            }
        });
        messageAlertButton.setIcon(messageAlertIcon);
        showMessageBar();
    }
    add(tabbedPane, BorderLayout.CENTER);          //初始化选项卡面板
    tabbedPane.setTabPlacement(SwingConstants.LEFT);
    ImageIcon userTicon = new ImageIcon(EQ.class
        .getResource("/image/tabIcon/tabLeft.PNG")); //创建用户列表图标
    tabbedPane.addTab(
        null,
        userTicon,
        createUserList(),
        "用户列表");                             //添加用户列表选项卡
    ImageIcon sysOTicon = new ImageIcon(EQ.class
        .getResource("/image/tabIcon/tabLeft2.PNG")); //创建系统操作图标
    tabbedPane.addTab(null, sysOTicon, createSysToolPanel(), "系统操作");
    //添加系统操作选项卡
}
```







```
ImageIcon sysSTicon = new ImageIcon(EQ.class
    .getResource("/image/tabIcon/tableft3.png")); //创建系统设置图标
tabbedPane.addTab(null, sysSTicon, createSysSetPanel(), "系统设置");
//添加系统设置选项卡
setAlwaysOnTop(true); //使窗体显示在最顶端
}
```

(3) 初始化 Socket 服务器, 指定端口使用 1111, 如果初始化失败, 提示用户服务器端口被占用, 或者本软件已经运行, 并退出程序。这个步骤很关键, 它用于接收其他用户发送的通信信息, 如果启动失败将无法接收信息, 所以必须退出系统。关键代码如下。

```
try { //启动通信服务端口
    ss = new DatagramSocket(1111); //初始化服务器
} catch (SocketException e2) {
    if (e2.getMessage().startsWith("Address already in use")) //如果异常提示该信息
        showMessageDialog("服务器端口被占用, 或者本软件已经运行。"); //显示提示信息
    System.exit(0); //退出系统
}
```

(4) 编写 checkPlacard() 方法, 该方法用于检测模块公告信息, 当公告路径中存在公告信息时, 该方法将从公告文件中获取完整信息, 然后调用 pushMessage() 方法将通告信息压入公告信息栈中。关键代码如下:

```
private void checkPlacard() { //检测公告信息方法
    String placardDir = preferences.get("placardPath", null); //获取公告路径
    if (placardDir == null) { //如果没有设置公告路径
        pushMessage("未设置公告路径"); //提示设置信息
        return;
    }
    File placard = new File(placardDir); //创建公告文件对象
    try {
        if (placard.exists() && placard.isFile()) {
            StringBuilder placardStr = new StringBuilder();
            Scanner sc = new Scanner(new FileInputStream(placard));
            while (sc.hasNextLine()) { //读取公告文件内容
                placardStr.append(sc.nextLine());
            }
            pushMessage(placardStr.toString()); //堆压公告信息
        }
    } catch (FileNotFoundException e) {
        pushMessage("公告路径错误, 或公告文件不存在"); //提示错误信息
    }
}
```

(5) 编写 initUserInfoButton() 方法, 该方法用于初始化本地用户信息, 并在主窗体左上角显示本地用户的头像和名称, 在用户更改本地用户名称时, 它会同步更新。关键代码如下。

```
private void initUserInfoButton() { //初始化用户信息按钮
    try {
        String ip = InetAddress.getLocalHost().getHostAddress(); //获取本地 ip
        User user = dao.getUser(ip); //从数据库获取用户对象
        userInfoButton.setIcon(user.getIconImg());
        userInfoButton.setText(user.getName());
        userInfoButton.setTextGap(JLabel.RIGHT); //设置文本显示在头像右侧
        userInfoButton.setToolTipText(user.getTipText()); //设置提示文本
        userInfoButton.getParent().doLayout();
    } catch (UnknownHostException el) {
        el.printStackTrace();
    }
}
```

(6) 编写 main() 方法。该方法是主程序的入口方法, 在该方法中首先获取用户设置的界面外观, 其中包括“当前系统”和“Java 默认”两种外观, 然后调用 UIManager 类的 setLookAndFeel() 方法设置指定的外观, 并生成主窗体对象。最后, 初始化服务器端口







和系统栏图标。关键代码如下：

```
public static void main(String args[]) {
    try {
        String laf = preferences.get("lookAndFeel", "java 默认");
        //获取用户选择的外观
        if (laf.indexOf("当前系统") > -1)
            UIManager.setLookAndFeel(UIManager
                .getSystemLookAndFeelClassName()); //设置外观
        EQ frame = new EQ(); //创建主窗体对象
        frame.setVisible(true); //显示窗体
        frame.SystemTrayInitial(); //初始化系统栏
        frame.server(); //启动服务端
        frame.checkPlacard(); //检测系统公告
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 7.3.3 记录窗体位置

(1) 为窗体添加控件监听器，当窗体改变大小或者移动位置时，调用 saveLocation() 方法将窗体的当前位置和大小保存到数据库中。关键代码如下：

```
addComponentListener(new ComponentAdapter() {
    public void componentResized(final ComponentEvent e) { //当窗体改变大小时
        saveLocation(); //保存改变到数据库
    }
    public void componentMoved(final ComponentEvent e) { //当窗体改变位置时
        saveLocation(); //保存改变到数据库
    }
});
```

(2) 编写 saveLocation() 方法，在该方法中调用 Dao 数据库操作类的 updateLocation() 方法将窗体位置和窗体大小保存到数据库中。关键代码如下：

```
private void saveLocation() { //保存主窗体位置的方法
    location = getBounds(); //获取窗体位置和大小
    dao.updateLocation(location); //调用 updateLocation() 方法
}
```

## 7.4 实现系统托盘

### 7.4.1 功能概述

系统托盘模块用于定义系统栏图标。局域网通信模块的主窗体是继承对话框窗体编写的，该窗体在系统任务栏不会显示相应的任务标题，如果主窗体最小化之后将会隐藏，这时必须使用快捷键或者系统托盘中的图标执行显示窗体的命令。本模块在系统托盘中的效果如图 7.8 所示。

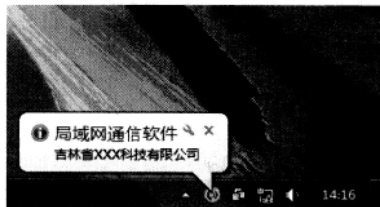


图 7.8 系统托盘效果





## 7.4.2 初始化系统托盘

在程序主类中编写 `SystemTrayInitial()` 方法，该方法用于初始化系统托盘。在方法中初始化系统托盘中的提示文本、系统栏图标，然后调用 `createMenu()` 方法为系统栏图标创建弹出菜单，同时为系统栏图标添加 `SysTrayActionListener` 类事件监听器。关键代码如下：

```
private void SystemTrayInitial() {
    if (!SystemTray.isSupported())           //判断当前系统是否支持系统栏
        return;
    try {
        String title = "局域网通信软件";      //系统栏图标提示文本的标题
        String company = "吉林省×××科技有限公司"; //系统栏图标提示文本
        SystemTray sysTray = SystemTray.getSystemTray(); //获取系统托盘对象
        Image image = Toolkit.getDefaultToolkit().getImage(
            EQ.class.getResource("/icons/sysTray.png")); //系统栏图标
        trayicon = new TrayIcon(image, title + "\n" + company, createMenu());
        //创建系统栏图标对象
        trayicon.setImageAutoSize(true);      //设置自动大小
        trayicon.addActionListener(new SysTrayActionListener()); //添加事件监听器
        sysTray.add(trayicon);                //添加系统栏图标到系统托盘
        trayicon.displayMessage(title, company, MessageType.INFO); //显示系统栏图标
        提示文本
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

## 7.4.3 实现弹出菜单

编写 `createMenu()` 方法，该方法用于创建系统栏图标的弹出菜单，该菜单包括“打开”、“访问服务器”和“退出”3个菜单项和菜单项分隔符。当用户选择“打开”菜单项时，将显示局域网通信模块的主窗体，在选择“访问服务器”菜单项之后将打开公共程序资源。另外主窗体没有提供退出功能，单击右上角的关闭按钮时执行的是最小化操作，所以只能使用“退出”菜单项完成程序的退出功能。关键代码如下：

```
private PopupMenu createMenu() { //创建系统栏菜单的方法
    PopupMenu menu = new PopupMenu();
    MenuItem exitItem = new MenuItem("退出");
    exitItem.addActionListener(new ActionListener() { //系统栏退出事件
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });
    MenuItem openItem = new MenuItem("打开");
    openItem.addActionListener(new ActionListener() { //系统栏打开菜单项事件
        public void actionPerformed(ActionEvent e) {
            if (!isVisible()) { //如果主窗体隐藏
                setVisible(true); //显示主窗体
                toFront(); //使主窗体显示在最上层
            } else
                toFront();
        }
    });
    MenuItem publicItem = new MenuItem("访问服务器"); //访问服务器菜单项事件
    publicItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

```







```
String serverPaeh = preferences.get("pubPath", null);
if (serverPaeh == null) {
    pushMessage("未设置公共程序路径"); //添加提示信息
    return;
}
Resource.startFolder(serverPaeh); //调用 startFolder() 方法
});
menu.add(publicItem); //添加访问服务器菜单项
menu.add(openItem); //添加打开菜单项
menu.addSeparator(); //添加菜单项分隔符
menu.add(exitItem); //添加退出菜单项
return menu;
}
```

### 7.4.4 双击托盘图标显示主窗体

创建 SysTrayActionListener 内部类，它实现了 ActionListener 接口，是系统栏图标的双击事件监听器，在用户双击系统栏图标之后，该监听器将实现主窗体的显示，这与系统栏图标的“打开”菜单项所实现的功能相同，但是双击系统栏图标会更加方便。关键代码如下：

```
class SysTrayActionListener implements ActionListener { //系统栏双击事件
    public void actionPerformed(ActionEvent e) {
        setVisible(true);
        toFront();
    }
}
```

## 7.5 实现系统工具

### 7.5.1 功能概述

局域网通信模块的系统工具模块起到维护作用，包括用户搜索、更换程序外观、系统升级 3 个功能。在局域网通信模块第一次运行时，用户搜索功能可以搜索内部网络中所有正在运行的计算机，并使用计算机的信息创建用户对象，然后将该用户对象保存到数据库中。在局域网通信模块有新版本程序时，可以使用系统升级功能直接升级到最新版本，而不用重新安装。系统工具模块的运行效果如图 7.9 所示。

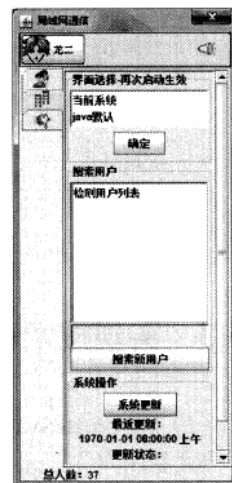


图 7.9 系统工具模块运行效果



## 7.5.2 实现界面选择

(1) 在程序主类中编写 `createSysToolPanel()` 方法, 用于创建系统工具选项卡, 在该选项卡中包括界面选择、用户搜索和系统操作 3 部分, 其中系统操作用于程序更新, 它们都被添加到系统工具面板中, `createSysToolPanel()` 方法必须设置好该面板的布局和初始化工作。关键代码如下:

```
private JScrollPane createSysToolPanel() {
    JPanel sysToolPanel = new JPanel(); //系统工具面板
    sysToolPanel.setLayout(new BorderLayout()); //设置面板布局
    JScrollPane sysToolScrollPanel = new JScrollPane(); //创建滚动面板
    sysToolScrollPanel
        .setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
    sysToolScrollPanel.setBorder(new EmptyBorder(0, 0, 0, 0)); //设置滚动面板的边框
    sysToolScrollPanel.setViewportView(sysToolPanel); //使面板可以滚动
    //设置系统工具面板边框
    sysToolPanel.setBorder(new BevelBorder(BevelBorder.LOWERED));
    ...//省略部分代码
    return sysToolScrollPanel;
}
```

(2) 在 `createSysToolPanel()` 方法中创建界面选择部分, 用于以列表控件显示两种外观选择, 当用户选择其中一种外观并单击“确定”按钮之后, 选择的外观会保存到首选项中, 然后提示用户重新运行本软件。关键代码如下:

```
JPanel interfacePanel = new JPanel(); //创建界面面板
sysToolPanel.add(interfacePanel, BorderLayout.NORTH); //设置面板的布局位置
interfacePanel.setLayout(new BorderLayout()); //设置面板布局管理器
//设置面板的 Title 边框
interfacePanel.setBorder(new TitledBorder("界面选择-再次启动生效"));
faceList = new JList(new String[]{"当前系统", "java 默认"}); //创建界面选择列表
interfacePanel.add(faceList);
faceList.setBorder(new BevelBorder(BevelBorder.LOWERED)); //设置列表的边框
final JPanel interfaceSubPanel = new JPanel();
interfaceSubPanel.setLayout(new FlowLayout());
interfacePanel.add(interfaceSubPanel, BorderLayout.SOUTH);
selectInterfaceOKButton = new JButton("确定"); //创建“确定”按钮
//添加按钮事件监听器
selectInterfaceOKButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        preferences.put("lookAndFeel", faceList.getSelectedValue()
            .toString()); //保存界面选择到首选项
        //提示重新运行软件
        JOptionPane.showMessageDialog(EQ.this, "重新运行本软件后生效");
    }
});
interfaceSubPanel.add(selectInterfaceOKButton); //添加按钮到面板中
```

## 7.5.3 实现搜索新用户

(1) 在 `createSysToolPanel()` 方法中创建用户搜索部分, 包括搜索列表、搜索进度条和“搜索新用户”按钮 3 个控件。当单击“搜索新用户”按钮时, 局域网通信模块会根据用户在系统设置界面所设置的 IP 搜索范围搜索所有计算机信息, 并创建对应的用户对象, 然后保存到数据库中。关键代码如下:

```
JPanel searchUserPanel = new JPanel(); //用户搜索面板
```







```

sysToolPanel.add(searchUserPanel);
searchUserPanel.setLayout(new BorderLayout()); //设置面板布局管理器
final JPanel searchControlPanel = new JPanel();
searchControlPanel.setLayout(new GridLayout(0, 1));
searchUserPanel.add(searchControlPanel, BorderLayout.SOUTH);
//定义搜索用户列表
final JList searchUserList = new JList(new String[]{"检测用户列表"});
final JScrollPane scrollPane_2 = new JScrollPane(searchUserList);
scrollPane_2.setDoubleBuffered(true);
searchUserPanel.add(scrollPane_2);
//设置用户列表边框
searchUserList.setBorder(new BevelBorder(BevelBorder.LOWERED));
searchUserButton = new JToggleButton("搜索新用户"); //创建“搜索新用户”按钮
searchUserButton.addActionListener(new
SearchUserActionListener(searchUserList)); //添加按钮的事件监听器
searchControlPanel.add(progressBar);
searchControlPanel.add(searchUserButton);
searchUserPanel.setBorder(new TitledBorder("搜索用户")); //设置面板的 Title 边框

```

(2) 创建“搜索新用户”按钮的事件监听器 SearchUserActionListener 类, 在该监听器中调用 Resource 工具类的 searchUsers() 方法搜索指定 IP 范围内的所有用户计算机信息。为避免等待搜索用户的业务逻辑过长而导致程序界面死锁, 监听器创建了另一个线程来执行用户搜索的业务。关键代码如下:

```

class SearchUserActionListener implements ActionListener {
    private final JList list;
    SearchUserActionListener(JList list) {
        this.list = list;
    }
    public void actionPerformed(ActionEvent e) {
        if (searchUserButton.isSelected()) { //如果单击“搜索新用户”按钮
            searchUserButton.setText("停止搜索"); //将其变成“停止搜索”按钮
            new Thread(
                new Runnable() { //创建内部线程
                    public void run() {
                        Resource.searchUsers(chatTree, progressBar,
                            list, searchUserButton); //调用 searchUsers() 方法
                    }
                }).start();
        } else
            searchUserButton.setText("搜索新用户"); //恢复按钮文本
    }
}

```

## 7.5.4 进行系统操作

(1) 在 createSysToolPanel() 方法中创建系统操作部分, 该部分包括“系统更新”按钮和显示程序更新信息的标签控件。当用户单击“系统更新”按钮时, 该按钮的事件监听器会下载当前最新版本的程序, 然后更新相应的标签控件提示用户更新信息。关键代码如下:

```

final JPanel sysUpdatePanel = new JPanel(); //系统更新面板
sysUpdatePanel.setOpaque(false);
sysUpdatePanel.setLayout(new GridBagLayout()); //设置面板布局管理器
sysUpdatePanel.setBorder(new TitledBorder("系统操作")); //设置边框
sysToolPanel.add(sysUpdatePanel, BorderLayout.SOUTH);
final JButton sysUpdateButton = new JButton("系统更新"); //定义“系统更新”按钮
final GridBagConstraints gridBagConstraints_1 = new GridBagConstraints();
//定义按钮布局
gridBagConstraints_1.gridx = 0;

```







```

gridBagConstraints_1.gridy = 0;
sysUpdatePanel.add(sysUpdateButton, gridBagConstraints_1);
sysUpdateButton.addActionListener(new SysUpdateListener()); //添加系统更新事件
final JLabel updateLabel = new JLabel("最近更新: ");
final GridBagConstraints updateLabelLayout = new GridBagConstraints();
updateLabelLayout.gridy = 1;
updateLabelLayout.gridx = 0;
sysUpdatePanel.add(updateLabel, updateLabelLayout);
final JLabel updateDateLabel = new JLabel(); //程序更新日期标签
Date date = new Date(localFile.lastModified());
String dateStr = String.format("%tF %<tr", date);
updateDateLabel.setText(dateStr);
final GridBagConstraints updateDateLayout = new GridBagConstraints();
updateDateLayout.gridy = 2;
updateDateLayout.gridx = 0;
sysUpdatePanel.add(updateDateLabel, updateDateLayout);
final JLabel updateStaticLabel = new JLabel("更新状态: ");
final GridBagConstraints updateStaticLayout = new GridBagConstraints();
updateStaticLayout.gridy = 3;
updateStaticLayout.gridx = 0;
sysUpdatePanel.add(updateStaticLabel, updateStaticLayout);
final JLabel updateInfoLabel = new JLabel(); //版本信息标签
checkSysInfo(updateInfoLabel); //调用检测版本更新的方法
final GridBagConstraints gridBagConstraints_5 = new GridBagConstraints();
gridBagConstraints_5.gridy = 4;
gridBagConstraints_5.gridx = 0;
sysUpdatePanel.add(updateInfoLabel, gridBagConstraints_5);
JPanel statePanel = new JPanel();
add(statePanel, BorderLayout.SOUTH);
statePanel.setLayout(new BorderLayout());
statePanel.add(stateLabel);
stateLabel.setText("总人数: " + chatTree.getRowCount()); //设置状态栏标签内容

```

(2) 创建“系统更新”按钮的事件监听器 SysUpdateListener 类, 该监听器在用户单击“系统更新”按钮时, 调用 Resource 工具类的 loginPublic()方法登录升级服务器, 然后调用 updateProject()方法, 更新局域网通信模块程序文件。关键代码如下:

```

//系统更新事件监听器
private final class SysUpdateListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        String username = preferences.get("username", null); //获取用户名
        String password = preferences.get("password", null); //获取密码
        if (username == null || password == null) { //如果用户名或密码为空
            pushMessage("未设置登录升级服务器的用户名或密码"); //提示设置登录信息
            return;
        }
        Resource.loginPublic(username, password); //调用 loginPublic() 方法
        updateProject(); //调用 updateProject() 方法
    }
}

```

(3) 编写更新程序的 updateProject()方法, 在该方法中分别创建本地程序和服务器最新程序文件对象, 然后解析两个文件的最后修改日期, 如果服务器程序的最后修改日期比本地程序更新, 那么获取服务器文件的内容并更新到本地文件中。关键代码如下:

```

private void updateProject() { //程序更新的方法
    netFilePath = preferences.get("updatePath", "EQ.jar"); //获取升级路径
    if (netFilePath.equals("EQ.jar")) { //如果首选项没有升级路径
        pushMessage("未设置升级路径"); //提示错误信息
        return;
    }
    netFile = new File(netFilePath); //创建服务器程序文件对象
    localFile = new File(user_dir + File.separator + "EQ.jar");
    //创建本地程序文件对象
}

```







```

if (localFile != null && netFile != null && netFile.exists()
    && localFile.exists()) {
    Date netDate = new Date(netFile.lastModified()); //提取服务器文件修改时间
    //提取本地程序最近修改时间
    Date localDate = new Date(localFile.lastModified());
    if (netDate.after(localDate)) { //如果服务器文件较新
        new Thread(new Runnable() {
            public void run() {
                try {
                    Dialog frameUpdate = new UpdateFrame(); //创建更新窗口
                    frameUpdate.setVisible(true); //显示更新窗口
                    Thread.sleep(2000); //线程休眠 2 秒
                    //创建文件输入流
                    FileInputStream fis = new FileInputStream(netFile);
                    //创建文件输出流
                    FileOutputStream fout = new FileOutputStream(
                        localFile);
                    int len = fis.available(); //计算文件有效长度
                    if (len > 0) {
                        byte[] data = new byte[len];
                        if (fis.read(data) > 0) { //如果读取到网络文件数据
                            fout.write(data); //写入本地文件中
                        }
                    }
                    fis.close(); //关闭文件输入流
                    fout.close(); //关闭文件输出流
                    frameUpdate.setVisible(false);
                    frameUpdate = null;
                    showMessageDialog("更新完毕, 请重新启动程序。");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    } else {
        showMessageDialog("已经是最新的程序了。");
    }
}
}

```

## 7.6 用户管理

### 7.6.1 功能概述

用户管理类似聊天软件的好友列表, 其中包含所有用户信息, 另外在用户名称上单击鼠标右键, 会弹出管理菜单, 菜单中包括“更名”、“添加用户”、“删除用户”、“信使群发”、“访问主机资源”和“访问公共程序”, 其中“访问主机资源”是访问该用户的共享文件夹。用户管理的运行效果如图 7.10 所示。



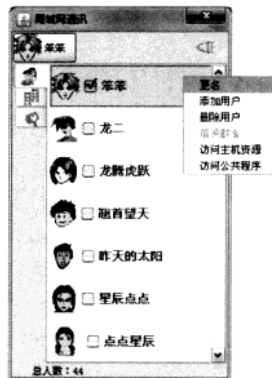


图 7.10 用户管理界面效果

## 7.6.2 创建用户树列表

(1) 创建 UserTreeRanderer 类, 该类继承 JPanel 类成为一个面板控件, 同时该类也实现了 TreeCellRenderer 接口成为树节点的渲染器。该类的构造方法中接收 3 个图标参数, 分别用于树节点的打开、关闭和叶节点的图标。关键代码如下:

```
public class UserTreeRanderer extends JPanel implements TreeCellRenderer {
    private Icon openIcon, closedIcon, leafIcon;           //节点图标
    private final JCheckBox label = new JCheckBox();         //用户选择图标
    private final JLabel headImg= new JLabel();             //用户头像
    private static User user;                               //用户对象
    public UserTreeRanderer() {
        super();
        user = null;
    }
    public UserTreeRanderer(Icon open, Icon closed, Icon leaf) {
        openIcon = open;                                   //节点展开图标
        closedIcon = closed;                               //节点折叠图标
        leafIcon = leaf;                                   //叶节点图标
        setBackground(new Color(0xF5B9BF));                //设置背景色
        label.setFont(new Font("宋体", Font.BOLD, 14));   //设置字体
        //选择用户的图标
        URL trueUrl = EQ.class.getResource
        ("/image/chexkBoxImg/CheckBoxTrue.png");
        label.setSelectedIcon(new ImageIcon(trueUrl));
        URL falseUrl = EQ.class.getResource
        ("/image/chexkBoxImg/CheckBoxFalse.png");          //取消用户的图标
        label.setIcon(new ImageIcon(falseUrl));
        label.setForeground(new Color(0, 64, 128));
        final BorderLayout borderLayout = new BorderLayout(); //创建布局管理器
        setLayout(borderLayout);                            //设置布局管理器
        user = null;
    }
    ...//省略部分代码
}
```

(2) 在 UserTreeRanderer 类中重写父类的 getTreeCellRendererComponent()方法, 它负责渲染树节点的界面样式。该方法将获取主窗体的宽度, 并使用该宽度值设置节点的宽度, 使节点与窗体同宽。当用户选择某个节点时, 该方法将使用指定颜色绘制节点的边框, 以突出该节点被选择的效果。关键代码如下:







```

public Component getTreeCellRendererComponent(JTree tree, Object value,
        boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)
{
    if (value instanceof DefaultMutableTreeNode) { //判断 value 是否是节点
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;
        Object uo = node.getUserObject(); //获取节点数据
        if (uo instanceof User) //如果该数据是 User 实例
            user = (User) uo; //初始化 User 对象
        } else if (value instanceof User) //如果 value 是 User 实例
            user = (User) value; //初始化 User 对象
        if (user != null && user.getIcon() != null) {
            int width = EQ.frame.getWidth(); //获取主窗体宽度
            if (width > 0)
                setPreferredSize(new Dimension(width, user.getIconImg().
                    getIconHeight())); //使节点与窗体同宽
            headImg.setIcon(user.getIconImg()); //设置用户头像
            tipText = user.getName();
        } else {
            if (expanded)
                headImg.setIcon(openIcon);
            else if (leaf)
                headImg.setIcon(leafIcon);
            else
                headImg.setIcon(closedIcon);
        }
        add(headImg, BorderLayout.WEST); //添加用户头像
        label.setText(value.toString()); //设置用户名称
        label.setOpaque(false);
        add(label, BorderLayout.CENTER); //添加用户名称
        if (selected) //如果该节点被选择
        {
            label.setSelected(true);
            //以指定颜色绘制边框
            setBorder(new LineBorder(new Color(0xD46D73), 2, false));
            setOpaque(true);
        } else //否则
        {
            setOpaque(false);
            label.setSelected(false);
            //恢复原来的边框颜色
            setBorder(new LineBorder(new Color(0xD46D73), 0, false));
        }
        return this;
    }
}

```

(3) 创建 ChatTree 类, 该类继承 JTree 类实现自定义的树控件, 并且使用了之前定义的 UserTreeRanderer 树节点渲染器, 它在构造方法中初始化类的属性, 然后调用 sortUsers() 方法添加并显示用户列表。关键代码如下:

```

public class ChatTree extends JTree {
    private DefaultMutableTreeNode root;
    private DefaultTreeModel treeModel;
    private List<User> userMap; //用户集合
    private Dao dao; //数据库操作类
    private EQ eq;
    public ChatTree(EQ eq) {
        super();
        root = new DefaultMutableTreeNode("root"); //初始化根节点
        treeModel = new DefaultTreeModel(root);
        userMap = new ArrayList<User>(); //初始化用户集合
        dao = Dao.getDao(); //初始化数据库操作类
        addMouseListener(new ThisMouseListener());
        setRowHeight(50); //设置节点的高度
        setToggleClickCount(2);
        setRootVisible(false); //隐藏根节点
        DefaultTreeCellRenderer defaultRanderer = new DefaultTreeCellRenderer();
        UserTreeRanderer treeRanderer = new UserTreeRanderer(defaultRanderer
            .getOpenIcon(), defaultRanderer.getClosedIcon(),

```





```
        defaultRenderer.getLeafIcon()); //创建自定义节点渲染器
setCellRenderer(treeRenderer); //设置该节点渲染器
setModel(treeModel);
sortUsers(); //添加并显示所有节点
this.eq = eq;
}
...//省略部分代码
}
```

### 7.6.3 在用户树中显示用户

在 ChatTree 类中编写 sortUsers() 方法，该方法的主体是一个内部线程，该线程首先获取本地 IP 地址，使用该地址从数据库中获取本地用户对象，并将本地用户显示在用户列表的首位。然后，从数据库中获取所有用户对象，将除自己以外的用户分别添加到用户列表中。最后，使第一个用户处于被选择的状态，并更新状态栏标签中显示的用户数量。关键代码如下：

```
private synchronized void sortUsers() {
    new Thread(new Runnable() {
        public void run() {
            try {
                Thread.sleep(100); //线程休眠 100 毫秒
                root.removeAllChildren();
                //获取本地 IP
                String ip = InetAddress.getLocalHost().getHostAddress();
                User localUser = dao.getUser(ip); //从数据库中获取自己
                if (localUser != null) {
                    DefaultMutableTreeNode node = new DefaultMutableTreeNode(
                        localUser);
                    root.add(node); //把自己显示在首位
                }
                userMap = dao getUsers(); //获取数据库中的所有用户
                Iterator<User> iterator = userMap.iterator();
                while (iterator.hasNext()) { //遍历用户集合
                    User user = iterator.next();
                    if (user.getIp().equals(localUser.getIp()))
                        continue;
                    root.add(new DefaultMutableTreeNode(user)); //添加用户到根节点
                }
                treeModel.reload();
                ChatTree.this.setSelectionRow(0); //使第一个节点被选择
                if (eq != null)
                    eq.setStatic(" 总人数: " + getRowCount()); //更新状态栏标签
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

### 7.6.4 从用户树中删除用户

在 ChatTree 类中编写 delUser() 方法，用于删除当前用户列表中选择的用户对象。该方法首先获取选择的树节点，从该节点中获取绑定的用户对象，然后用对话框提示用户是否确认删除，如果经过用户确认，将调用 delUser() 方法从数据库中删除用户信息，最后调







用根节点的 `remove()` 方法删除该用户节点。关键代码如下：

```
public void delUser() {
    TreePath path = getSelectionPath();           //获取被选择的树节点
    if (path == null)
        return;
    User user = (User) ((DefaultMutableTreeNode) path
        .getLastPathComponent()).getUserObject(); //获取节点中的用户对象
    int operation = JOptionPane.showConfirmDialog(this, "确定要删除用户: " + user
        + "?", "删除用户", JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);           //以对话框提示确认删除
    if (operation == JOptionPane.YES_OPTION) {
        dao.delUser(user);                         //调用 delUser() 方法
        root.remove((DefaultMutableTreeNode) path.getLastPathComponent());
        //删除该节点
        treeModel.reload();
    }
}
```

### 7.6.5 向用户树中添加用户

在 `ChatTree` 类中编写 `addUser()` 方法，它可以向用户列表中添加新用户。该方法首先使用传递的 IP 参数到数据库中获取对应的用户对象，如果成功获取用户对象，说明数据库已存在该 IP 地址的用户，软件将使用对话框提示“用户已存在”，否则执行该 IP 地址的搜索任务，当确定该 IP 地址可以访问之后，为该 IP 地址创建一个新的用户对象并添加到数据库中，然后调用 `sortUsers()` 方法重新加载用户列表并提示用户添加成功。关键代码如下：

```
public boolean addUser(String ip, String operation) {
    try {
        if (ip == null)
            return false;
        User oldUser = dao.getUser(ip);           //从数据库中获取 IP 相同的用户
        if (oldUser == null) {                     //如果数据库中不存在该用户
            InetAddress addr = InetAddress.getByName(ip);
            if (addr.isReachable(1500)) {          //如果该用户 IP 可以访问
                String host = addr.getHostName(); //获取它的主机名称
                User newUser = new User();         //创建新用户对象
                newUser.setIp(ip);                 //初始化用户 IP
                newUser.setHost(host);             //初始化用户主机名
                newUser.setName(host);             //设置用户姓名
                newUser.setIcon("1.gif");          //初始化用户头像
                dao.addUser(newUser);              //添加该用户到数据库
                sortUsers();                       //重新加载用户列表
                if (!operation.equals("search"))   //如果不是系统自动搜索
                    JOptionPane.showMessageDialog(EQ.frame, "用户" + host
                        + "添加成功", "添加用户",
                        JOptionPane.INFORMATION_MESSAGE); //弹出添加成功对话框
                return true;
            }
        }
        //如果该用户 IP 不可访问
        if (!operation.equals("search"))          //并且不是系统自动搜索
            //对话框提示错误
            JOptionPane.showMessageDialog(EQ.frame, "检测不到用户 IP: "
                + ip, "错误添加用户", JOptionPane.ERROR_MESSAGE);
        return false;
    }
}
//如果数据库存在该 IP 用户
//并且不是系统自动搜索
```





```

        //对话框提示用户已存在
        JOptionPane.showMessageDialog(EQ.frame, "已经存在用户 IP" + ip,
            "不能添加用户", JOptionPane.WARNING_MESSAGE);
        return false;
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

```

## 7.7 实现通信

### 7.7.1 功能概述

通信是局域网通信模块的核心模块，它用于不同职工之间的通信，这种通信方式能够实现多个职工之间的通话，而不存在类似电话的占线问题，增加了任务分配的新方式，从而提高企业的工作效率。该模块可以使用 UDP 协议和系统信使两种方式发送通信信息。通信界面运行效果如图 7.11 所示。

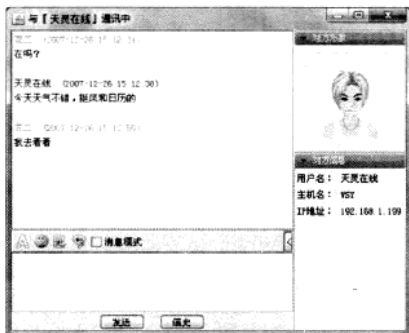


图 7.11 通信界面运行效果

### 7.7.2 实现通信窗体

(1) 创建 TelFrame 窗体类，并定义窗体需要的所有控件，例如，接收信息的文本框、输入发送信息的文本框、发送按钮、信使按钮及各种滚动面板等，另外还需要定义访问数据库的 Dao 实例，在通信窗口右侧显示的用户信息需要使用 Dao 实例从数据库中获取。关键代码如下：

```

public class TelFrame extends JFrame {
    private Dao dao; //数据库操作类
    private User user; //用户实例
    private JTextPane receiveText = new JTextPane(); //信息接收文本框
    private JScrollPane scrollPane = new JScrollPane(); //滚动面板
    private JTextPane sendText = new JTextPane(); //信息编辑文本框
    private JScrollPane scrollPane_1 = new JScrollPane(); //滚动面板
    private JSplitPane splitPane = new JSplitPane(); //分割面板
}

```







```

private JButton sendButton = new JButton(); // “发送”按钮
private final JButton messageButton = new JButton(); // “信使”按钮
private JPanel panel = new JPanel();
private final static Map<String, TelFrame> instance = new HashMap<String,
TelFrame>(); // 窗体实例集合
private final JCheckBox messageMode = new JCheckBox(); // 消息模式复选框
private JToolBar toolBar = new JToolBar(); // 工具栏
private JToggleButton toolFontButton = new JToggleButton(); // “字体”按钮
private JButton toolFaceButton = new JButton(); // “表情”按钮
private final JScrollPane scrollPane_2 = new JScrollPane(); // 滚动面板
private final JLabel label_1 = new JLabel();
private JPanel panel_3 = new JPanel();
private byte[] buf; // 数据缓冲
private DatagramSocket ss; // Socket
private String ip; // IP 地址
private DatagramPacket dp; // 数据报
private TelFrame frame; // 窗体实例
private ChatTree tree; // 用户列表实例
private int rightPanelWidth = 148; // 右侧面板宽度
...//省略部分代码
}

```

(2) 编写 getInstance() 方法, 用于获取唯一的窗体实例。该方法创建的所有窗体实例都会保存到 Map 集合类的实例中, 除非退出局域网通信模块, 否则窗体的实例对象会一直保存在这个集合类中, 并且用户再次打开已存在的窗体时, 将直接从集合类中获取, 不 recreated 新的窗体实例。关键代码如下:

```

public static synchronized TelFrame getInstance(DatagramSocket ssArg,
DatagramPacket dp, ChatTree treeArg) {
    String tmpIp = dp.getAddress().getHostAddress(); // 获取数据报的 IP 地址
    if (!instance.containsKey(tmpIp)) { // 如果集合中不存在该用户窗体
        TelFrame frame = new TelFrame(ssArg, dp, treeArg); // 创建窗体实例
        instance.put(tmpIp, frame); // 将窗体实例保存到集合中
        frame.receiveInfo(treeArg); // 接收信息
        if (!frame.isVisible()) { // 如果窗体处于隐藏状态
            frame.setVisible(true); // 显示窗体
        }
        frame.setState(JFrame.NORMAL);
        frame.toFront(); // 将窗体放置在最前端
        return frame;
    } else {
        TelFrame frame = instance.get(tmpIp); // 如果集合中包含该用户窗体
        frame.receiveInfo(treeArg); // 从集合中获取窗体实例
        if (!frame.isVisible()) { // 接收信息
            frame.setVisible(true); // 显示窗体
        }
        frame.setState(JFrame.NORMAL);
        frame.toFront();
        return frame;
    }
}

```

(3) 在构造方法中初始化 TelFrame 类的所有控件属性, 该构造方法接收 DatagramSocket、DatagramPacket 和 ChatTree 类的 3 个参数, 它们分别是数据 Socket 服务、数据报和用户列表的实例对象, 在通信窗体中需要使用它们。关键代码如下:

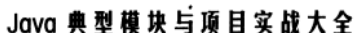
```

public TelFrame(DatagramSocket ssArg, DatagramPacket dpArg, final
ChatTree treeArg) {
    this.tree = treeArg; // 获取用户列表对象
    ip = dpArg.getAddress().getHostAddress(); // 获取数据报中的 IP 地址
    user = dao.getUser(ip); // 获取该 IP 的用户对象
    dao = Dao.getDao(); // 初始化 Dao 实例
    ss = ssArg; // 获取数据服务对象
    dp = dpArg; // 获取数据报
    buf = dp.getData(); // 获取数据报中的数据
    setBounds(200, 100, 521, 424); // 设置窗体位置和大小
    getContentPane().add(splitPane); // 添加分割面板
}

```

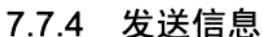






### 7.7.3 接收信息

```
private void receiveInfo(final ChatTree tree) { //接收信息
    if (buf.length > 0) {
        String rText = new String(buf).replace("" + (char) 0, "");
        String hostAddress = dp.getAddress().getHostAddress();
        String info = dao.getUser(hostAddress).getName();
        info = info + " (" + new Date().toLocaleString() + ")";
        appendReceiveText(info, Color.BLUE);
        appendReceiveText(rText + "\n", null);
    }
}
```



Java学习群: 72030155





```

class sendActionListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        String sendInfo = getSendInfo()           //获取发送信息
        if (sendInfo == null)
            return;
        insertUserInfoToReceiveText(tree);
        appendReceiveText(sendInfo + "\n", null);   //添加到信息文本框
        byte[] tmpBuf = sendInfo.getBytes();
        DatagramPacket tdp = null;                 //创建数据报
        try {
            tdp = new DatagramPacket(tmpBuf, tmpBuf.length,
                                     new InetSocketAddress(ip, 1111)); //初始化数据报
            ss.send(tdp);                          //发送数据报
        } catch (SocketException e2) {
            e2.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
            JOptionPane.showMessageDialog(TelFrame.this, e1
                                         .getMessage());
        }
        sendText.setText(null);                    //清空发送文本框
        sendText.requestFocus();                  //使发送文本框获得焦点
        if (messageMode.isSelected())              //如果选择了“消息模式”复选框
            setState(ICONIFIED);                  //则窗体最小化
    }
}

```

### 7.7.5 系统信使

创建 MessageButtonActionListener 内部类，该类是“信使”按钮的事件监听器。当用户输入通信信息，并单击“信使”按钮时，该事件监听器的 actionPerformed()方法会调用 Resource 工具类的 sendMessage()方法将通信信息以系统信使方式发送到对方计算机。关键代码如下：

```

private class MessageButtonActionListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        try {
            Document doc = sendText.getDocument(); //获取文档对象
            String sendInfo = doc.getText(0, doc.getLength()); //获取发送的通信信息
            if (sendInfo.equals("") || sendInfo == null) { //限制空信息的发送
                JOptionPane.showMessageDialog(TelFrame.this, "不能发送空信息。");
                return;
            }
            insertUserInfoToReceiveText(tree);
            appendReceiveText(sendInfo, null);
            //调用 sendMessage() 方法
            Resource.sendMessage(user, sendInfo, frame);
            sendText.setText(null); //清空发送信息文本框
            sendText.requestFocus(); //使文本框获得焦点
        } catch (BadLocationException e1) {
            e1.printStackTrace();
        }
    }
}

```



# 第 8 章

---

## 区域地图模块

(Swing+Java DB+绘图技术实现)

---

随着科技的发展，定位服务的应用越来越普及，例如，常见的车载GPS系统、智能手机的定位系统等，这些系统都是在已经绘制好的地图上标示用户的位置。此外，还提供了餐饮、娱乐等场所的查询功能。本模块将设计一个通用的区域地图，用户可以根据实际需要，将其建成各种功能的导航系统。通过本章的学习，读者能够学到：

- » 图片截取和缩放等操作
- » 维护树模型的方法
- » 万年历选择框的实现方法
- » 高级搜索的实现方法





## 8.1 区域地图模块概述

### 8.1.1 设计思路

区域地图模块将设计成为一个可重用的模块,软件开发人员只要将该模块添加到自己的系统中,就实现了区域地图的功能。在本模块中,主要提供了地图操作、标记操作和更换地图三大功能,其中更换地图功能用来设置该模块采用的地图,下面将详细介绍另外两大功能的设计思路。

地图操作包括缩放地图、移动地图、还原地图和鹰眼漫游功能,其中鹰眼漫游功能是用来辅助其他功能的。通过鹰眼漫游功能可以快速查看当前显示区域在整个地图中的位置。缩放功能用来调整地图的显示比例,缩放范围为4倍。移动功能用来调整地图的显示区域,通过缩放功能和移动功能,可以在不同比例下查看地图的任意区域。

标记操作包括维护类别、维护标记和搜索标记功能。为了便于管理,提供了以树结构管理类别的功能。在地图的任意位置都可以建立标记,并且可以查看、修改和删除现有标记。在不同缩放比例下建立的标记,仅当显示比例不小于标记时比例才可见。搜索标记分为常用搜索和高级搜索,常用搜索只能以标记文本为条件,并且将同时在标记名称和标记说明中搜索;高级搜索不仅可以设置标记文本的具体搜索范围,还可以将创建日期和标记类别作为搜索条件,满足搜索条件的标记将显示在搜索结果列表中。选中列表中的某一标记后,该标记将被描红显示在地图显示区域的中间,如果当前的查看比例未达到建立标记时的比例,则调整查看比例为建立标记时的比例。

### 8.1.2 功能结构

根据区域地图的设计思路,可以将该模块分为地图操作、标记操作和更换地图3部分来实现,具体的功能结构如图8.1所示。

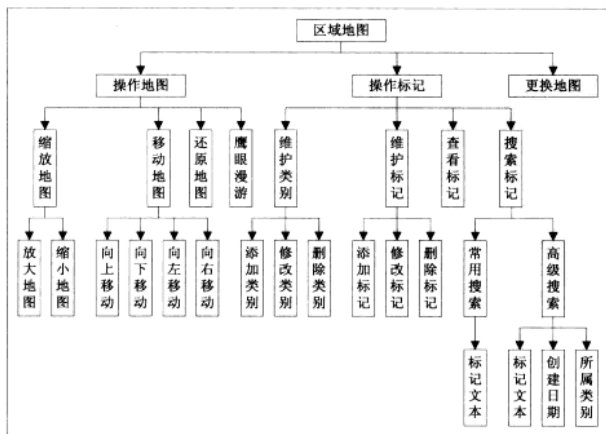


图 8.1 区域地图模块功能结构





### 8.1.3 程序预览

如图 8.2 所示的为区域地图模块的主界面效果，在该图中已经标出了窗体中各个部分的主要功能。

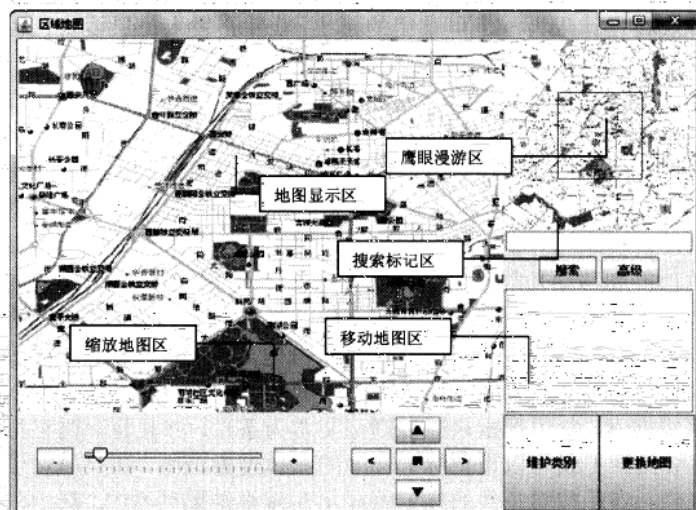


图 8.2 区域地图模块主界面

如图 8.3 所示为通过“广场”查询标记，并查看选中标记的具体位置。选中标记被描红并居中显示，如果当前显示比例小于标记时比例，则调整显示比例为建立标记时的比例。

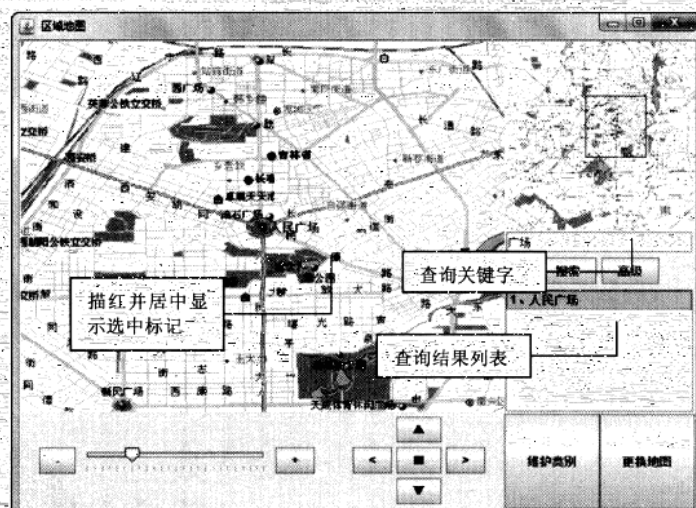


图 8.3 查询并查看标记

将光标移动到标记点上时，将变为手状，效果如图 8.4 所示。此时单击鼠标右键，可以查看和维护该标记点的信息。



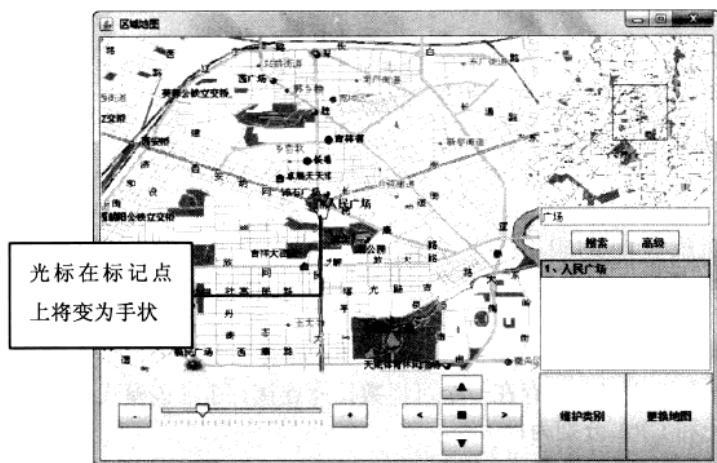


图 8.4 查看或维护标记信息

在地图主界面只提供了通过关键字查询标记的功能，单击“高级”按钮将打开如图 8.5 所示的“高级搜索”对话框，在这里可以设置关键字的搜索范围，还可以将创建日期和所属类别设为搜索条件。

如图 8.6 所示的对话框用来维护标记类别，标记类别支持树状结构，即在一个类别中还可以包含子类别。



图 8.5 “高级搜索”对话框



图 8.6 “维护类别”对话框

如图 8.7 所示的对话框用来创建标记，标记信息由名称、类型、日期和说明组成。在默认情况下为显示标记名称，如果不希望显示标记名称，可以选中对话框中的“否”单选按钮。

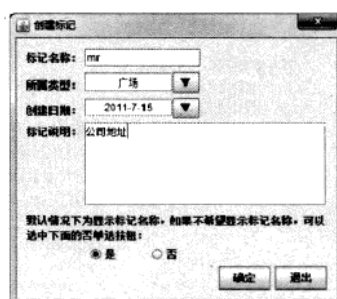


图 8.7 “创建标记”对话框





## 8.2 关键技术

### 8.2.1 Java DB 数据库

在 JDK6 中,集成了 Java DB,它是一个纯 Java 编写的数据库系统,虽然大小仅有 2MB,但是完全支持 SQL 语句、存储过程和触发器等。此外,还可以很方便地嵌入到 Java 桌面应用程序中。

区域地图模块中的标记类别信息和标记信息,就是使用该数据库保存的。这里采用内嵌模式的数据库,该模式仅允许存在一个可用数据库连接。通过该模式访问数据库时,数据库驱动类的路径和 URL 分别为:

```
private static final String DRIVERCLASS = "org.apache.derby.  
jdbc.EmbeddedDriver"; // 驱动  
private static final String URL = "jdbc:derby:db_map"; // 协议
```



数据库连接协议中的“db\_map”为欲连接数据库的名称。

JDBC 类的静态代码块负责加载数据库驱动,并判断数据库是否存在。如果不存在则创建数据库,并保存返回的数据库连接对象。其关键代码如下:

```
static { // 通过静态方法加载数据库驱动,并且在数据库不存在的情况下创建数据库  
    try {  
        Class.forName(DRIVERCLASS); // 加载数据库驱动  
        File databaseFile = new File("db_map"); // 创建数据库文件对象  
        if (!databaseFile.exists()) { // 判断数据库文件是否存在  
            String[] sqls = new String[5]; // 定义创建数据库的 SQL 语句  
            sqls[0] = "create table tb_map (id int not null,name varchar  
(8) not null)";  
            sqls[1] = "insert into tb_map(id,name) values(1,'map.jpg')";  
            sqls[2] = "create table tb_sort (id int not null,father_id int  
not null,name varchar(20) not null,  
primary key (id))";  
            sqls[3] = "create table tb_sign (id int not null,sort_id int not null,x  
int not null,y int not null,  
title varchar(20) not null,show int not  
null,scale float not null,  
date date not null,remark varchar(200),primary  
key (id))";  
            sqls[4] = "create view v_sign_sort as SELECT tb_sign.x, tb_sign.y,  
tb_sign.title, tb_sort.id,  
tb_sort.name, tb_sign.show, tb_sign.scale, tb_sign.date,  
tb_sign.remark  
FROM tb_sign INNER JOIN tb_sort ON tb_sign.sort_id =  
tb_sort.id ";  
            // 创建数据库连接  
            conn = DriverManager.getConnection(URL + ";create=true");  
            threadLocal.set(conn); // 保存数据库连接  
            Statement stmt = conn.createStatement(); // 创建数据库连接状态对象  
            for (int i = 0; i < sqls.length; i++) { // 通过执行 SQL 语句创建数据库  
                stmt.execute(sqls[i]); // 执行 SQL 语句  
            }  
            stmt.close(); // 关闭数据库连接状态对象  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```







```
}
}
```

getConnection()方法用于获得数据库连接,它从 ThreadLocal 类的对象中获得数据库连接对象,如果不存在则创建,该方法的关键代码如下:

```
protected static Connection getConnection() {           // 创建数据库连接的方法
    conn = threadLocal.get();                          // 从线程中获得数据库连接
    if (conn == null) {                                // 没有可用的数据库连接
        try {
            conn = DriverManager.getConnection(URL); // 创建新的数据库连接
            threadLocal.set(conn);                  // 将数据库连接保存到线程中
        } catch (Exception e) {
            String[] infos = { "未能成功连接数据库!", "请确认本软件是否已经运行!" };
            JOptionPane.showMessageDialog(null, infos); // 弹出连接数据库失败的提示
            System.exit(0);                             // 关闭系统
            e.printStackTrace();
        }
    }
    return conn;
}
```

closeConnection()方法用于关闭数据库连接,它从 ThreadLocal 类的对象中获得数据库连接对象,如果存在则关闭,该方法的关键代码如下:

```
protected static boolean closeConnection() {           // 关闭数据库连接的方法
    boolean isClosed = true;                          // 默认关闭成功
    conn = threadLocal.get();                          // 从线程中获得数据库连接
    threadLocal.set(null);                             // 清空线程中的数据库连接
    if (conn != null) {                                // 数据库连接可用
        try {
            conn.close();                             // 关闭数据库连接
        } catch (SQLException e) {
            isClosed = false;                          // 关闭失败
            e.printStackTrace();
        }
    }
    return isClosed;
}
```

## 8.2.2 万年历选择框技术

在创建和修改标记时,日期是使用日期选择框设置的。单击日期文本框后面的按钮,将在文本框下方弹出日期选择框,效果如图 8.8 所示。



图 8.8 选择日期对话框



217





### 1. 向表格中添加按钮控件

表格是选择日期对话框的主体，用来将指定的月份按星期进行显示。参考 Windows 系统中的日历控件，将表格设置成 6 行 7 列。如果本月的 1 号不是星期一，则用上个月的最后几天填充本月的开始。为了便于区分，将字体设置为灰色。通常情况下字体为黑色，如果为星期六、星期日或当日，字体将采用特殊颜色。

除了需要设置不同的字体颜色，还要捕获单元格单击事件。这里并不是直接添加日期到表格单元格中，而是将日期设置到按钮控件中，然后将按钮控件添加到表格中。其中灰色字体是将按钮控件设置为不可用。因为添加到表格单元格中的是按钮控件，所以在实现表格模型时需要重写 `getColumnClass(int columnIndex)` 方法，返回表格单元格中值的类型为按钮控件类型，表格模型类 `MTableModel` 的代码如下：

```
public class MTableModel extends AbstractTableModel {
    private static final long serialVersionUID = -3750844363213456807L;
    private final String[] columnNames;           // 表格列名数组
    private final Object[][] tableDatas;          // 表格数据数组
    public MTableModel(String[] columnNames, Object[][] tableDatas) {
        super();
        this.columnNames = columnNames;
        this.tableDatas = tableDatas;
    }
    @Override
    public int getRowCount() {                    // 返回表格行数
        return tableDatas.length;
    }
    @Override
    public int getColumnCount() {                 // 返回表格列数
        return columnNames.length;
    }
    @Override
    // 返回指定单元格的值
    public Object getValueAt(int rowIndex, int columnIndex) {
        return tableDatas[rowIndex][columnIndex];
    }
    @Override
    public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
        // 设置指定单元格的值
        tableDatas[rowIndex][columnIndex] = aValue;
    }
    @Override
    public String getColumnName(int column) {     // 返回指定列的名称
        return columnNames[column];
    }
    @Override
    public Class<?> getColumnClass(int columnIndex) { // 返回指定列值的类型
        return JButton.class;                    // 为按钮组件类型
    }
}
```



说明 `getColumnClass()` 方法返回值为 `JButton.class`，说明表格单元格中添加的是按

钮控件。

添加到表格单元格中的按钮控件，将不绘制边框，控件的背景色为白色，文本和边框间的距离为 0。它通过 `Mbutton` 类创建，其关键代码如下：

```
public class MButton extends JButton {
    private static final long serialVersionUID = -3780740398880605358L;
    public MButton(int day) {
```





```

        super(day + "");
        setBorderPainted(false);           // 不绘制边框
        setBackground(Color.WHITE);        // 背景色为白色
        setMargin(new Insets(0, 0, 0, 0)); // 文本和边框间的距离为 0
        setFont(new Font("宋体", Font.BOLD, 14)); // 设置字体及样式
    }
}

```

下面重写表格的单元格对象，这是通过实现 TableCellRenderer 接口中的 getTableCellRendererComponent() 方法完成的。用来定义表格单元格的 MTableCell 类的代码如下：

```

public class MTableCell extends JPanel implements TableCellRenderer {
    private static final long serialVersionUID = 2843371869359375263L;
    private static String selectedDay;           // 被选中的日期
    public MTableCell() {
        setLayout(new BorderLayout());           // 采用边框式布局
    }
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected,                               boolean hasFocus, int row, int
        column) {
        JButton button = (JButton) value;           // 获得单元格中的按钮对象
        if (hasFocus && button.isEnabled()) {        // 如果选中了该按钮
            selectedDay = button.getText();           // 获得日期
        }
        removeAll();                                 // 移除其他按钮组件
        add(button, BorderLayout.CENTER);             // 添加该按钮组件到边框式布局的中心
        return this;                                 // 返回该单元格对象
    }
    public static String getSelectedDay() {           // 返回选中的日期
        return selectedDay;
    }
    public static void setSelectedDay(String selectedDay) { // 设置选中的日期
        MTableCell.selectedDay = selectedDay;
    }
}

```

在创建表格时，需要设置表格模型，以及默认的表格单元格对象，其关键代码如下：

```

// 定义列名数组
String[] columnNames = { "一", "二", "三", "四", "五", "六", "日" };
Object[][] tableDatas = new Object[6][7];           // 定义数据数组
tableModel = new MTableModel(columnNames, tableDatas); // 创建表格模型对象
table.setModel(tableModel);                         // 为表格设置表格模型
table.setRowHeight(20);                             // 设置表格的行高
table.setRowSelectionAllowed(false);                 // 不允许选中表格行
// 设置默认的表格单元格
table.setDefaultRenderer(JButton.class, new MTableCell());
JTableHeader tableHeader = table.getTableHeader(); // 获得表格头对象
tableHeader.setFont(new Font("", Font.BOLD, 12));   // 定义表格头字体
tableHeader.setBackground(Color.ORANGE);            // 定义表格头的背景色
scrollPane.setViewportView(table);                     // 添加表格到滚动面板

```

initTableModel() 方法负责初始化表格模型。首先解析本月一号为星期几，然后计算本星期空出的天数，所有表示空出天数的按钮都不可用。然后添加本月的各天，并根据是否为特殊日期（如周六、周日和当前日期）设置不同的前景色。最后通过添加下个月的日期填满表格，所添加的按钮均不可用。该方法的关键代码如下：

```

private void initTableModel() {
    int row = 0;           // 行索引
    int col = 0;           // 列索引
    MButton button = null; // 声明一个按钮对象
}

```







```
try {
    dateFormat.parse(year + "-" + month + "-1"); // 解析当月1号
} catch (ParseException e) {
    e.printStackTrace();
}
// 获得 Calendar 类的对象
Calendar firstDayOfMonth = dateFormat.getCalendar();
// 获得为一周的第几天
int dayOfWeek = firstDayOfMonth.get(Calendar.DAY_OF_WEEK) - 1;
if (dayOfWeek == 0) { // 为周日
    dayOfWeek = 7; // 按中国习惯放在最后
}
// 上个月
int lastMonthDays = 31; // 默认上个月为 31 天
if (month > 2) {
    lastMonthDays = daysOfMonth[month - 1]; // 获得上个月的具体天数
}
// 用上个月的最后几天填充本周的开始几天
for (int day = lastMonthDays - dayOfWeek + 2; day <= lastMonthDays; day++) {
    button = new MButton(day); // 创建按钮对象
    button.setEnabled(false); // 设置按钮不可用
    tableModel.setValueAt(button, row, col); // 设置到表格模型的相应位置
    // 计算索引值
    if (col == 6) {
        row++;
        col = 0;
    } else {
        col++;
    }
}
// 当月
for (int day = 1; day <= daysOfMonth[month]; day++) {
    button = new MButton(day); // 创建按钮对象
    if (col > 4) {
        if (col == 5) { // 为星期六
            button.setForeground(Color.GREEN); // 设置按钮的前景色
        } else { // 为星期日
            button.setForeground(Color.RED); // 设置按钮的前景色
        }
    }
    if (day == DAY) {
        if (year == YEAR && month == MONTH) { // 为当天
            button.setForeground(Color.ORANGE); // 设置按钮的前景色
        }
    }
    tableModel.setValueAt(button, row, col); // 设置到表格模型的相应位置
    // 计算索引值
    if (col == 6) {
        row++;
        col = 0;
    } else {
        col++;
    }
}
// 下个月
int nextMonthDays = 42 - (row * 7 + col); // 计算需要的天数
// 用下个月的开始几天填充表格的剩余单元格
for (int day = 1; day <= nextMonthDays; day++) {
    button = new MButton(day); // 创建按钮对象
    button.setEnabled(false); // 设置按钮不可用
    tableModel.setValueAt(button, row, col); // 设置到表格模型的相应位置
    // 计算索引值
    if (col == 6) {
        row++;
        col = 0;
    } else {
        col++;
    }
}
```







```

        col++;
    }
}

```



**注意** 使用 MTableModel 类的 setValueAt() 方法为表格模型设置单元格的值时, 设置

的是 Mbutton 类的对象。

## 2. 在文本框下方显示对话框

单击日期文本框后面的按钮, 将在文本框下方弹出日期选择框。buttonActionPerformed() 方法负责处理该事件, 主要思路是通过文本框在屏幕上的起始绘制点和它的首选大小, 来计算日期选择框的显示位置, 其关键代码如下:

```

private void buttonActionPerformed(java.awt.event.ActionEvent evt) {
    Dimension textFieldPreferredSize = getTextField().getPreferredSize();
    // 获得文本框的首选大小
    // 获得文本框在屏幕上的起始绘制点
    Point locationOnScreen = getTextField().getLocationOnScreen();
    int x = (int) locationOnScreen.getX(); // 定义对话框在X轴的起始绘制点
    int y = (int) (locationOnScreen.getY() + textFieldPreferredSize
    .getHeight()); // 定义对话框在Y轴的起始绘制点
    dialog = new CalendarDialog(null, true); // 创建对话框对象
    // 获得对话框的首选大小
    Dimension dialogPreferredSize = dialog.getPreferredSize();
    dialog.setBounds(x, y, (int) dialogPreferredSize.getWidth(), (int)
    dialogPreferredSize.getHeight());
    new Thread() { // 创建并开启一个线程
        @Override
        public void run() { // 重构该方法
            while (true) {
                if (MTableCell.getSelectedDay() != null) { // 如果用户选择了日期
                    getTextField().setText(CalendarDialog.getYear() + "-" + Calendar
                    Dialog.getMonth() + "-" +
                    MTableCell.getSelectedDay());
                    dialog.dispose(); // 销毁日期选择框
                    MTableCell.setSelectedDay(null);
                    break; // 跳出循环
                }
                try {
                    Thread.sleep(1000); // 休眠1秒
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }.start();
    dialog.setVisible(true); // 设置日期选择框可见
}

```



**说明** 通过 getLocationOnScreen() 方法, 可以获得控件起始绘制点在屏幕中的坐标,

该方法将返回一个 Point 类型的对象。通过 Point 类的 getX() 方法可以获得在 X 轴的坐标值, 通过 Point 类的 getY() 方法可以获得在 Y 轴的坐标值。



221





### 8.2.3 滑块控件使用技术

在开发地图比例调整功能时，将使用滑块控件，该控件通常用于选取一组连续值，其运行效果如图 8.9 所示。

当移动滑块时，滑块的值将在最大值和最小值之间变化。通过为滑块控件添加 `ChangeListener` 事件监听器，可以捕获到滑块值的变化，通过 `getValue()` 方法可以得到滑块当前的值。

滑块的主标尺和副标尺是相互独立的，即在定义标尺间隔时可以随意设置，副标尺的间隔并不需要参考主标尺的间隔，例如，这里是将主标尺的间隔设置为 50，副标尺的间隔设置为 25，也可以将副标尺的间隔设置为 30，但是这样会显得标尺上很乱，效果如图 8.10 所示，所以建议副标尺的间隔要能够被主标尺的间隔整除。

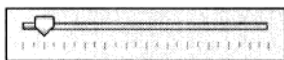


图 8.9 用来调整比例的滑块控件

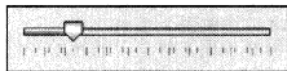


图 8.10 效果很乱的标尺

如果执行 `setSnapToTicks()` 方法，并将参数设置为 `true`，则当移动滑块至标记之间时，滑块将自动吸附到距离最近的标记上。通过这一功能可以确定滑块的取值间隔，默认为不自动吸附到距离最近的标记上。

`JSlider` 类提供的常用方法如表 8.1 所示。

表 8.1 `JSlider` 类中的常用方法

参 数	说 明
<code>setPaintTrack(boolean b)</code>	设为 <code>false</code> 表示不绘制滑道，默认为绘制
<code>setPaintTicks(boolean b)</code>	设为 <code>true</code> 表示绘制标尺，默认为不绘制
<code>setPaintLabels(boolean b)</code>	设为 <code>true</code> 表示绘制标签，默认为不绘制
<code>setInverted(boolean b)</code>	设为 <code>true</code> 表示反向绘制标尺
<code>setSnapToTicks(boolean b)</code>	设为 <code>true</code> 表示自动吸附至最近的标记，默认为不吸附
<code>setOrientation(int orientation)</code>	设为 <code>VERTICAL</code> 表示垂直绘制滑块控件，默认为水平绘制
<code>setMaximum(int maximum)</code>	设置滑块的最大值
<code>setMinimum(int minimum)</code>	设置滑块的最小值
<code>setMajorTickSpacing(int n)</code>	设置主标尺的间距
<code>setMinorTickSpacing(int n)</code>	设置副标尺的间距
<code>setValue(int n)</code>	设置滑块的当前值
<code>getValue()</code>	获得滑块的当前值
<code>addChangeListener(ChangeListener l)</code>	为滑块控件添加 <code>ChangeListener</code> 事件监听器
<code>removeChangeListener(ChangeListener l)</code>	移除指定的 <code>ChangeListener</code> 事件监听器







## 8.2.4 列表控件使用

在开发搜索标记功能时将利用列表控件显示搜索结果，效果如图 8.11 所示。



图 8.11 用来显示搜索结果的列表控件

用来定义该列表控件的代码如下：

```
searchResultList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
searchResultList.setFixedCellWidth(20);
searchResultList.addListSelectionListener(new
    javax.swing.event.ListSelectionListener() {
        @Override
        public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
            searchResultListValueChanged(evt);
        }
    });
searchResultScrollPane.setViewportViewView(searchResultList);
```



通常将列表控件添加到滚动窗格中，否则如果列表无法显示所有选项，将无法令这些选项可见。

对于列表控件中的元素，可以使用 JList 组件，他共有三种选取模式，设置方法为通过 JList 类的 setSelectionMode(int selectionMode)方法，该方法的入口参数可以通过 ListSelectionModel 类中的静态常量设置。这三种选取模式包括两种多选模式和一种单选模式，一种多选模式是只能连续选取多个，通过静态常量 SINGLE\_INTERVAL\_SELECTION（常量值为 1）设置，选取效果如图 8.12 所示。

另一种多选模式是既能连续选取多个，又能间隔选取多个，通过静态常量 MULTIPLE\_INTERVAL\_SELECTION（常量值为 2）设置，选取效果如图 8.13 所示。

本系统采用的是单选模式，单选模式即只能选取选项列表中的一项，通过静态常量 SINGLE\_SELECTION（常量值为 0）设置，选取效果如图 8.14 所示。

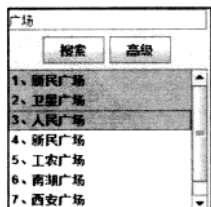


图 8.12 多选模式（必须连选）



图 8.13 多选模式（连隔均可）

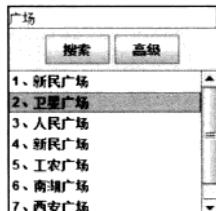


图 8.14 单选模式





JList 类提供的常用方法如表 8.2 所示。

表 8.2 JList 类中的常用方法

参 数	说 明
setSelectedIndex(int index)	选中指定索引的一个选项
setSelectedIndices(int[] indices)	选中指定索引的一组选项
getSelectedIndices()	以 int[] 形式获得被选中的所有选项的索引值
getSelectedValues()	以 Object[] 形式获得被选中的所有选项的内容
clearSelection()	取消所有被选中的项
isSelectionEmpty()	查看是否有被选中的项，如果有则返回 true
ensureIndexIsVisible(int index)	使指定项在选择窗口中可见
setFixedCellWidth(int width)	设置选择窗口中每个选项的宽度
setFixedCellHeight(int height)	设置选择窗口中每个选项的高度
setVisibleRowCount(int visibleRowCount)	设置在选择窗口中最多可见选项的个数
getPreferredScrollableViewportSize()	获得使指定个数的选项可见需要的窗口高度
setSelectionMode(int selectionMode)	设置列表框的选择模式，即单选还是多选
addListSelectionListener(ListSelectionListener listener)	为列表框添加 ListSelectionListener 监听器

## 8.2.5 维护树模型

标记分类功能是通过树控件实现的，效果如图 8.15 所示，通过单击工具栏中的按钮，可以维护标记类别。

树中的信息是通过树模型进行维护的。如果树模型发生变化，一定要重新加载相应的部分，否则维护信息可能无法同步到树中。下面就详细介绍维护树模型的方法。

addButtonActionPerformed() 方法负责处理“添加类别”按钮事件。如果存在选中的类别，则添加为该类别的子类别，否则添加为顶级类别。该方法的关键代码如下：



图 8.15 “维护类别”对话框

```
private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    MTreeNode lastSelectedNode = (MTreeNode) sortTree.  
getLastSelectedPathComponent(); // 获得选中节点对象  
    int fatherId = (lastSelectedNode == null ? 0 : lastSelectedNode.getId());  
    // 定义父类别 ID  
    String albumName = getAlbumName(fatherId, "添加类别", "请输入类别名称:");  
    // 获得添加类别名称  
    if (albumName != null) {  
        int id = dao.insertSort(fatherId, albumName); // 输入了类别名称  
        // 添加到数据库  
        if (lastSelectedNode == null) { // 未选中任何节点  
            MTreeNode root = (MTreeNode) treeModel.getRoot(); // 获得树的根节点  
            root.add(new MTreeNode(id, albumName)); // 添加子节点  
            treeModel.reload(); // 重新加载树模型  
        }  
    }  
}
```







```

    } else {
        // 获得选中节点路径对象
        TreePath selectionPath = sortTree.getSelectionPath();
        if (!sortTree.isExpanded(selectionPath)) { // 该节点尚未展开
            sortTree.expandPath(selectionPath); // 展开节点
        } else { // 该节点已经展开
            lastSelectedNode.add(new MTreeNode(id, albumName)); // 添加子节点
            treeModel.reload(lastSelectedNode); // 重新加载指定节点
        }
    }
}
}

```

updButtonActionPerformed()方法负责处理“修改类别”按钮事件, 如果不存在被选中的类别, 则弹出提示框, 该方法的关键代码如下:

```

private void updButtonActionPerformed(java.awt.event.ActionEvent evt) {
    MTreeNode lastSelectedNode = (MTreeNode) sortTree.
    getLastSelectedPathComponent(); // 获得选中节点对象
    if (lastSelectedNode == null) { // 未选中任何节点
        JOptionPane.showMessageDialog(this, "请选择要修改的类别!", "友情提示",
        JOptionPane.INFORMATION_MESSAGE); // 弹出提示框
    } else { // 存在选中的节点
        String albumName = getAlbumName(lastSelectedNode.getId(), "修改类别", "
        请输入类别" + lastSelectedNode.getUserObject() + "的新名称: ");
        // 获得修改后的名称
        if (albumName != null) { // 输入了类别名称
            dao.updateSortNameById(lastSelectedNode.getId(), albumName);
            // 保存到数据库
            lastSelectedNode.setUserObject(albumName); // 修改节点的用户标签
            treeModel.reload(lastSelectedNode); // 重新加载该节点
        }
    }
}
}

```



说明 方法 setUserObject(Object userObject)用来设置树节点的用户标签。

delButtonActionPerformed()方法负责处理“删除类别”按钮事件, 如果不存在被选中的类别, 则弹出需要先选中类别的提示框; 否则弹出确认删除的提示框, 避免因为失误操作而删除类别, 该方法的关键代码如下:

```

private void delButtonActionPerformed(java.awt.event.ActionEvent evt) {
    MTreeNode lastSelectedNode = (MTreeNode) sortTree.
    getLastSelectedPathComponent(); // 获得选中节点对象
    if (lastSelectedNode == null) { // 未选中任何节点
        JOptionPane.showMessageDialog(this, "请选择要删除的类别!", "友情提示",
        JOptionPane.INFORMATION_MESSAGE); // 弹出提示框
    } else { // 存在选中的节点
        String[] infos = { "如果删除该类别, 将同时删除其包含的", "所有类别, 以及属于这
        些类别的标记!", "", "确定要删除类别" + lastSelectedNode.getUserObject() + "吗? " };
        // 定义确认删除提示信息
        // 弹出提示框
        int i = JOptionPane.showConfirmDialog(this, infos, "友情提示",
        JOptionPane.YES_NO_OPTION);
        if (i == 0) { // 确认删除
            dao.deleteSortById(lastSelectedNode.getId()); // 从数据库中删除
            list.setModel(new javax.swing.AbstractListModel() {
                @Override
                public int getSize() {
                    return 0;
                }
            });
        }
    }
}
}

```





```
@Override
public Object getElementAt(int i) {
    return null;
}

});
slider.setValue(slider.getValue() + 1); // 清空查询结果列表
// 触发滑块事件
MTreeNode parentNode = (MTreeNode) lastSelectedNode.getParent();
// 获得删除节点的父节点
parentNode.remove(lastSelectedNode); // 移除该节点
treeModel.reload(parentNode); // 重新加载父节点
}
```

## 8.3 地图处理器

### 8.3.1 功能概述

因为一张地图的尺寸要比地图显示区的尺寸大很多,所以每次只能从地图上截取一部分进行显示。在截取地图时需要考虑 3 种因素,分别为地图的缩放比例、地图的视觉中心点和地图显示区的大小。MapProcessor 类会根据这 3 种因素进行一系列的计算,然后将通过截取得到的部分地图缩放后返回,效果如图 8.16 所示。

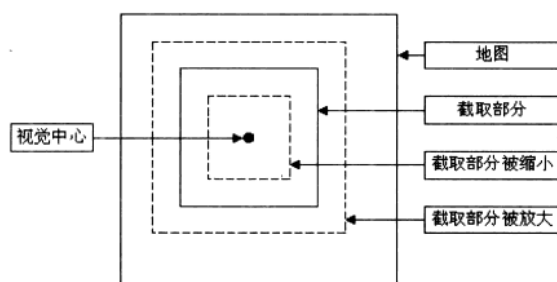


图 8.16 地图的截取与缩放

### 8.3.2 获得小地图

在 MapProcessor 类中定义了若干属性,用来保存截取地图时需要考虑的因素及地图对象,其具体定义如下:

```
private BufferedImage map; // 地图对象
private int viewportWidth; // 地图显示区的宽度
private int viewportHeight; // 地图显示区的高度
private int showCenterX; // 地图视觉中心的水平坐标
private int showCenterY; // 地图视觉中心的垂直坐标
private int cutMapWidth; // 截取地图的宽度
private int cutMapHeight; // 截取地图的高度
private float scale; // 缩放比例
```







cut()方法利用这些属性从地图中截取小地图,并将其缩放为地图显示区域大小,该方法的关键代码如下:

```
private ImageIcon cut() {
    BufferedImage subimage = map.getSubimage(
        showCenterX - cutMapWidth / 2, showCenterY - cutMapHeight
        / 2, // 开始截取点的坐标
        cutMapWidth, cutMapHeight); // 截取地图的大小
    // 返回当前显示的地图
    return new ImageIcon(subimage.getScaledInstance(viewportWidth,
        viewportHeight, Image.SCALE_DEFAULT));
}
```

在调整地图显示比例时,需要使用 adjustScale()方法获得显示的小地图。当地图的显示比例发生变化时,将影响到截图地图的大小。如果是调小比例,还可能影响到地图的视觉中心。因为当前显示的小地图位于地图的某一边缘,调小比例将增大截取地图的尺寸。adjustScale()方法的关键代码如下:

```
public ImageIcon adjustScale(int scale) {
    float forestallScale = this.scale; // 调整前的比例
    dealWithScale(scale); // 处理比例
    dealWithCutMapSize(); // 处理需要截取地图的大小
    if (this.scale < forestallScale) { // 比例缩小,地图的视觉中心可能变化
        validateShowCenterX(); // 验证地图视觉中心的水平坐标
        validateShowCenterY(); // 验证地图视觉中心的垂直坐标
    }
    return this.cut(); // 返回当前显示的地图
}
```



**说明** adjustScale()方法中并没有真实的业务逻辑代码,只是调用了一些方法,这些

方法在后面将进行详细讲解。

当在水平或垂直方向移动地图时,需要通过 moveOfHorizontal()方法或 moveOfVertical()方法获得将要显示到地图显示区的小地图,这两个方法的参数均为移动的距离。如果当前显示的小地图被放大或缩小,则应该相应地缩小或放大移动的距离,它们的关键代码如下:

```
public ImageIcon moveOfHorizontal(int distance) {
    if (scale == 0) { // 不缩放
        this.showCenterX += distance;
    } else {
        if (scale > 0) { // 放大
            this.showCenterX += distance / scale;
        } else { // 缩小
            this.showCenterX += distance * -scale;
        }
    }
    validateShowCenterX(); // 验证地图视觉中心的水平坐标
    return this.cut(); // 返回当前显示的地图
}

public ImageIcon moveOfVertical(int distance) {
    if (scale == 0) { // 不缩放
        this.showCenterY += distance;
    } else {
        if (scale > 0) { // 放大
            this.showCenterY += distance / scale;
        } else { // 缩小
            this.showCenterY += distance * -scale;
        }
    }
}
```





```
        }  
        validateShowCenterY();  
        return this.cut();  
    }  
    // 验证地图视觉中心的垂直坐标  
    // 返回当前显示的地图
```

当调整窗体大小时,会影响地图显示区域的大小。此时需要使用 `adjustWindow()` 方法获得将要显示的小地图。该方法的参数依次为调整后地图显示区域的高和宽。其关键代码如下:

```
public ImageIcon adjustWindow(int viewportWidth, int viewportHeight) {  
    this.viewportWidth = viewportWidth;    // 设置视口宽度  
    this.viewportHeight = viewportHeight;  // 设置视口高度  
    dealWithCutMapSize();                  // 处理需要截取地图的大小  
    validateShowCenterX();                 // 验证地图视觉中心的水平坐标  
    validateShowCenterY();                 // 验证地图视觉中心的垂直坐标  
    return this.cut();                     // 返回当前显示的地图  
}
```

`revert()` 方法用于还原地图的视觉中心,视觉中心默认为地图的中心,该方法的关键代码如下:

```
public void revert() {  
    this.showCenterX = map.getWidth() / 2;    // 设置地图视觉中心的水平坐标  
    this.showCenterY = map.getHeight() / 2;   // 设置地图视觉中心的垂直坐标  
}
```

### 8.3.3 处理缩放和显示位置

上面的代码中曾多次调用 `dealWithScale()`、`dealWithCutMapSize()`、`validateShowCenterX()` 和 `validateShowCenterY()` 几个方法,下面将详细讲解它们的具体功能。

`dealWithScale()` 方法用来处理缩放比例。因为滑块控件的值为 `int` 型,且缩放范围为在原图大小的基础上放大或缩小 4 倍,所以需要通过该方法将滑块值解析为缩放比例,地图的缩放比例与滑块值的对应关系如图 8.17 所示。正数表示在原图大小的基础上放大,负数表示在原图大小的基础上缩小,0 表示不缩放。

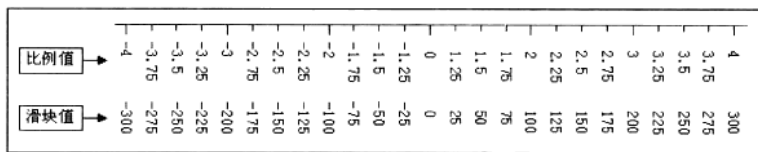


图 8.17 缩放比例与对应的滑块值

`dealWithScale()` 方法的关键代码如下:

```
private void dealWithScale(int scale) {  
    if (scale == 0) {  
        this.scale = 0;    // 不缩放  
    } else {  
        this.scale = scale / 100f;  
        if (scale > 0) {  
            this.scale += 1;    // 放大  
        } else {  
            this.scale -= 1;    // 缩小  
        }  
    }  
}
```







dealWithCutMapSize()方法用于处理截取地图的大小,影响截取地图大小的因素为缩放比例和地图显示区的大小,该方法的关键代码如下:

```
private void dealWithCutMapSize() {
    if (scale == 0) { // 不缩放
        cutMapWidth = viewportWidth;
        cutMapHeight = viewportHeight;
    } else {
        if (scale > 0) { // 放大
            cutMapWidth = (int) (viewportWidth / scale);
            cutMapHeight = (int) (viewportHeight / scale);
        } else { // 缩小
            cutMapWidth = (int) (viewportWidth * -scale);
            cutMapHeight = (int) (viewportHeight * -scale);
        }
    }
}
```

validateShowCenterX()方法用于验证地图视觉中心的水平坐标是否有效,该方法的关键代码如下:

```
private void validateShowCenterX() {
    int w = cutMapWidth / 2;
    if (cutMapWidth % 2 != 0) {
        w += 1;
    }
    if (showCenterX < w) { // 从地图左边缘开始截取
        showCenterX = w;
    } else {
        if (showCenterX > map.getWidth() - w) { // 截取至地图右边缘
            showCenterX = map.getWidth() - w;
        }
    }
}
```

validateShowCenterY()方法用于验证地图视觉中心的垂直坐标是否有效,该方法的关键代码如下:

```
private void validateShowCenterY() {
    int h = cutMapHeight / 2;
    if (cutMapHeight % 2 != 0) {
        h += 1;
    }
    if (showCenterY < h) { // 从地图上边缘开始截取
        showCenterY = h;
    } else {
        if (showCenterY > map.getHeight() - h) { // 截取至地图下边缘
            showCenterY = map.getHeight() - h;
        }
    }
}
```

## 8.4 地图显示

### 8.4.1 功能概述

在系统中共有两个地图,其中一个是为了具体操作的大地图,另一个是为了鹰眼漫游的小地图。因为它们的功能不相同,所以用来绘制这两个地图的标签控件也不相同,其显





示效果如图 8.18 和 8.19 所示。

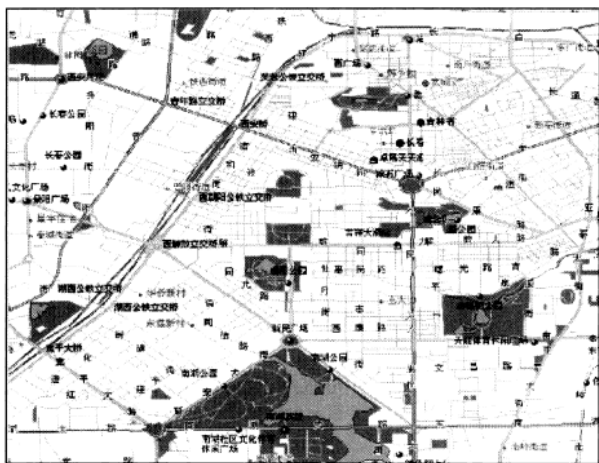


图 8.18 大地图显示效果

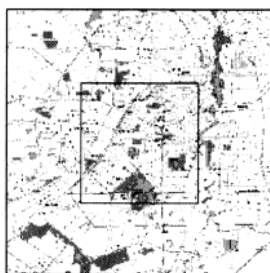


图 8.19 小地图显示效果

## 8.4.2 绘制大地图

大地图由 `BigMapLabel` 类负责绘制, 内容包括通过截取得到的小地图及满足条件的标记点, 如图 8.18 所示。在该类中定义了如下几个用来控制标记点绘制的属性:

```
private static final Dao dao = Dao.getInstance();  
private static final Vector<Integer> ids = new Vector();           // 标记主键集  
private static final Vector<Integer> xs = new Vector();           // X 轴坐标集  
private static final Vector<Integer> ys = new Vector();           // Y 轴坐标集  
private static final Vector<String> texts = new Vector();         // 标记文本集  
private static final Vector<Boolean> shows = new Vector();        // 是否显示集  
private static MapProcessor mapProcessor;                          // 地图处理器对象  
private static int operateIndex = -1;                              // 操作标记点索引  
private static int stressId = -1;                                  // 着重标记点主键
```

接着重写了 `paintComponent()` 方法, 用于将标记点绘制到地图上。它是一个以标记点坐标为圆心, 半径为 5 像素的圆点。如果绘制文本则在标记点右侧。在绘制过程中, 需要判断该标记点是否为着重标记点, 如果是则绘制为红色, 否则绘制为橘黄色。该方法的关键代码如下:

```
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    g.setColor(Color.ORANGE);                                     // 设置笔触颜色  
    for (int i = 0; i < texts.size(); i++) {  
        int x = xs.get(i);                                       // 标记点的 X 轴坐标  
        int y = ys.get(i);                                       // 标记点的 Y 轴坐标  
        if (ids.get(i) == stressId) {                             // 为着重标记点  
            g.setColor(Color.RED);                                // 修改笔触颜色  
            g.fillOval(x - 5, y - 5, 10, 10);                   // 绘制标记点  
            if (shows.get(i)) {                                   // 如果显示标记文本  
                g.drawString(texts.get(i), x + 8, y + 5);        // 绘制标记文本  
            }  
            g.setColor(Color.ORANGE);                             // 还原笔触颜色  
        } else {                                                  // 为普通标记点  
            g.fillOval(x - 5, y - 5, 10, 10);                   // 绘制标记点  
        }  
    }  
}
```







```

        if (shows.get(i)) { // 如果显示标记文本
            g.drawString(texts.get(i), x + 8, y + 5); // 绘制标记文本
        }
    }
}

```

此外,还重写了 setIcon()方法,用于在每次更新小地图时,同步更新需要绘制的标记点信息。具体的更新操作由 refreshSigns()方法完成,这里需要将标记点在地图上的坐标转换为在小地图上的坐标。这两个方法的关键代码如下:

```

public void setIcon(Icon icon) {
    if (mapProcessor == null) { // 尚未获得地图处理器对象
        if (InstancePool.getMapProcessor() != null) { // 如果地图处理器对象已经创建
            mapProcessor = InstancePool.getMapProcessor(); // 获得地图处理器对象
            refreshSigns(); // 刷新标记
        }
    } else { // 已经获得地图处理器对象
        refreshSigns(); // 刷新标记
    }
    super.setIcon(icon);
}

private void refreshSigns() {
    ids.clear();
    xs.clear();
    ys.clear();
    texts.clear();
    shows.clear();
    // 查询符合绘制条件的标记
    Vector signs = dao.selectShowSigns(mapProcessor.getShowCenterX(),
        mapProcessor.getShowCenterY(),
        mapProcessor.getCutMapWidth(), mapProcessor.getCutMapHeight(),
        mapProcessor.getScale());
    // 计算相对原点的坐标
    int originX = mapProcessor.getShowCenterX() - mapProcessor.getCutMapWidth()
    / 2;
    int originY = mapProcessor.getShowCenterY() - mapProcessor.getCutMapHeight()
    / 2;
    float scale = mapProcessor.getScale();
    for (int i = 0; i < signs.size(); i++) { // 遍历标记点集合
        Vector sign = (Vector) signs.get(i); // 获得标记点
        // 计算相对坐标
        int x = (Integer) sign.get(2) - originX;
        int y = (Integer) sign.get(3) - originY;
        if (scale != 0) { // 进行了缩放
            if (scale > 0) { // 放大
                x = (int) (x * scale);
                y = (int) (y * scale);
            } else { // 缩小
                x = (int) (x / -scale);
                y = (int) (y / -scale);
            }
        }
        addSign((Integer) sign.get(0), x, y, sign.get(4).toString(), ((Integer)
            sign.get(5) == 1 ? true : false));
    }
}

```



**说明** 一定要先更新需要绘制的标记点信息,然后再调用父类方法设置图片。

isEnteredSign(int x, int y)方法用来查看在指定坐标点是否存在标记点,如果存在则返回 true,否则返回 false,其关键代码如下:







```

public boolean isEnteredSign(int x, int y) {
    int xDistance, yDistance; // 距标记点的距离
    for (int i = 0; i < texts.size(); i++) {
        xDistance = Math.abs(x - xs.get(i)); // 计算水平距离
        yDistance = Math.abs(y - ys.get(i)); // 计算垂直距离
        if (xDistance < 5 && yDistance < 5) { // 光标在该标记点之上
            operateIndex = i; // 保存该标记点的索引
            mapProcessor.rightClick(xs.get(i), ys.get(i)); // 保存该标记点的坐标
            return true; // 光标在该标记点之上
        }
    }
    mapProcessor.rightClick(x, y); // 保存该标记点的坐标
    return false; // 光标未在任何标记点之上
}

```

### 8.4.3 绘制小地图

小地图由 `SmallMapLabel` 类负责绘制，内容包括通过缩小得到的小地图及一个矩形框。该矩形框内的区域为大地图显示的区域，如图 8.19 所示。在该类中定义了如下两个属性，分别为小地图的水平缩小比例和垂直缩小比例。

```

private float xScale; // 鹰眼漫游图的水平缩小比例
private float yScale; // 鹰眼漫游图的垂直缩小比例

```

这里重写了 `paintComponent()` 方法，用于绘制矩形框，绘制的前提条件是已经创建了 `MapProcessor` 类对象，其关键代码如下：

```

protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (mapProcessor == null) {
        mapProcessor = InstancePool.getMapProcessor();
        if (mapProcessor != null) {
            refreshScale(); // 计算鹰眼漫游图的缩小比例
            drawRect(g); // 绘制矩形框
        }
    } else {
        drawRect(g); // 绘制矩形框
    }
}

```

在 `paintComponent()` 方法中调用了 `refreshScale()` 方法，该方法用来计算鹰眼漫游图的水平缩小比例和垂直缩小比例，其关键代码如下：

```

public void refreshScale() {
    // 计算鹰眼漫游图的水平缩小比例
    xScale = 200f / mapProcessor.getMap().getWidth();
    // 计算鹰眼漫游图的垂直缩小比例
    yScale = 200f / mapProcessor.getMap().getHeight();
}

```

在 `paintComponent()` 方法中还调用了 `drawRect()` 方法，该方法用来绘制矩形框。首先计算矩形框的位置，然后要验证是否合理，最后才进行绘制，该方法的关键代码如下：

```

private void drawRect(Graphics g) {
    int w = (int) (xScale * mapProcessor.getCutMapWidth()); // 定义矩形宽度
    int h = (int) (yScale * mapProcessor.getCutMapHeight()); // 定义矩形高度
    // 定义水平轴的起始绘制坐标
    int x = (int) (xScale * mapProcessor.getShowCenterX()) - w / 2;
    int y = (int) (yScale * mapProcessor.getShowCenterY()) - h / 2; // 定义垂直轴的起始绘制坐标
    // 验证水平坐标
    if (x < 0) {

```







```

        x = 0;
    } else {
        if (x + w == 200) {
            x -= 1;
        }
    }
    // 验证垂直坐标
    if (y < 0) {
        y = 0;
    } else {
        if (y + h == 200) {
            y -= 1;
        }
    }
    g.setColor(Color.RED);
    g.drawRect(x, y, w, h);
}
// 设置笔触颜色
// 绘制矩形框

```

## 8.5 地图操作

### 8.5.1 功能概述

对地图的操作包括缩放地图、移动地图和还原地图,通过配合使用缩放功能和移动功能,可以在不同比例下查看地图的任意区域,通过还原功能可以迅速地将地图恢复到最初的位置和比例。操作地图的控件如图 8.20 所示。

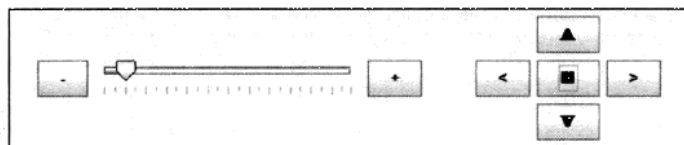


图 8.20 操作地图控件

### 8.5.2 实现地图缩放功能

缩放功能用来调整地图的显示比例,通过一个滑块和两个按钮来实现,如图 8.20 所示。利用滑块可以大跨度地调整地图的缩放比例,利用按钮可以对缩放比例进行微调,每单击一次可以调整一个最小单位,即滑块标尺中的一个小单元格。最大可将地图放大到原图大小的 4 倍,最小可将地图缩小为原图大小的四分之一。

```

private void sliderStateChanged(javax.swing.event.ChangeEvent evt) {
    int v = slider.getValue();
    // 判断比例调整按钮是否可用
    if (v == slider.getMinimum()) {
        subButton.setEnabled(false);
        if (!addButton.isEnabled()) {
            addButton.setEnabled(true);
        }
    }
    // 调整为最小值
    // 调小按钮不可用
    // 如果调大按钮不可用
    // 设置调大按钮可用
    } else if (v == slider.getMaximum()) {
        addButton.setEnabled(false);
        if (!subButton.isEnabled()) {
            subButton.setEnabled(true);
        }
    }
    // 调整为最大值
    // 调大按钮不可用
    // 如果调小按钮不可用
}

```



233





```
        subButton.setEnabled(true); // 设置调小按钮可用
    }
    } else {
        if (!subButton.isEnabled()) { // 如果调小按钮不可用
            subButton.setEnabled(true); // 设置调小按钮可用
        }
        if (!addButton.isEnabled()) { // 如果调大按钮不可用
            addButton.setEnabled(true); // 设置调大按钮可用
        }
    }
    // 刷新地图
    bigMapLabel.setIcon(mapProcessor.adjustScale(v)); // 重绘地图显示区
    SwingUtilities.updateComponentTreeUI(smallMapLabel); // 重绘鹰眼漫游区
    // 判断水平移动按钮是否可用
    int w = mapProcessor.getCutMapWidth() / 2;
    if (mapProcessor.getShowCenterX() - w < 2 || mapProcessor.getMap().getWidth()
    - mapProcessor.getShowCenterX() - w < 2) {
        if (mapProcessor.getShowCenterX() <= w) {
            leftButton.setEnabled(false);
        } else {
            rightButton.setEnabled(false);
        }
    } else {
        if (!leftButton.isEnabled()) {
            leftButton.setEnabled(true);
        }
        if (!rightButton.isEnabled()) {
            rightButton.setEnabled(true);
        }
    }
    // 判断垂直移动按钮是否可用
    int h = mapProcessor.getCutMapHeight() / 2;
    if (mapProcessor.getShowCenterY() - h < 2 || mapProcessor.getMap()
    .getHeight() - mapProcessor.getShowCenterY() - h < 2) {
        if (mapProcessor.getShowCenterY() <= h) {
            upButton.setEnabled(false);
        } else {
            downButton.setEnabled(false);
        }
    } else {
        if (!upButton.isEnabled()) {
            upButton.setEnabled(true);
        }
        if (!downButton.isEnabled()) {
            downButton.setEnabled(true);
        }
    }
}
}
```

单击滑块左侧和右侧按钮时，分别触发 `subButtonActionPerformed()` 方法和 `addButtonActionPerformed()` 方法。在这两个方法中只是相应的修改了滑块的值，这一动作也将发出 `ChangeEvent` 事件，它们的关键代码如下：

```
private void subButtonActionPerformed(java.awt.event.ActionEvent evt) {
    slider.setValue(slider.getValue() - 25); // 将缩放比例减小一个单位
}
private void addButtonActionPerformed(java.awt.event.ActionEvent evt) {
    slider.setValue(slider.getValue() + 25); // 将缩放比例增加一个单位
}
```





### 8.5.3 实现地图移动功能

移动功能利用4个按钮来调整地图的显示位置,如图8.20所示。每单击一次按钮可以向相应的方向移动100像素。当移动到地图某一边缘时,相应的按钮将不可用。

当向上或向下移动地图时,将首先刷新地图显示区域,然后判断按钮是否可用,最后刷新鹰眼漫游区域,负责处理这两个按钮动作事件的方法关键代码如下:

```
private void upButtonActionPerformed(java.awt.event.ActionEvent evt) {
    bigMapLabel.setIcon(mapProcessor.moveOfVertical(-100)); // 刷新地图显示区
    // 判断垂直移动按钮是否可用
    if (!downButton.isEnabled()) {
        downButton.setEnabled(true);
    } else {
        if (mapProcessor.getShowCenterY() - mapProcessor.getCutMapHeight() / 2
        < 2) {
            upButton.setEnabled(false);
        }
    }
    SwingUtilities.updateComponentTreeUI(smallMapLabel); // 刷新鹰眼漫游区
}
private void downButtonActionPerformed(java.awt.event.ActionEvent evt) {
    getBigMapLabel().setIcon(mapProcessor.moveOfVertical(100)); // 刷新地图显示区
    if (!upButton.isEnabled()) {
        upButton.setEnabled(true);
    } else {
        if (mapProcessor.getMap().getHeight() - mapProcessor.getShowCenterY() -
        mapProcessor.getCutMapHeight() / 2 < 2) {
            downButton.setEnabled(false);
        }
    }
    SwingUtilities.updateComponentTreeUI(smallMapLabel); // 刷新鹰眼漫游区
}
```

当向左或向右移动地图时,将首先刷新地图显示区域,然后判断按钮是否可用,最后刷新鹰眼漫游区域,负责处理这两个按钮动作事件的方法关键代码如下:

```
private void leftButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 刷新地图显示区
    getBigMapLabel().setIcon(mapProcessor.moveOfHorizontal(-100));
    // 判断水平移动按钮是否可用
    if (!rightButton.isEnabled()) {
        rightButton.setEnabled(true);
    } else {
        if (mapProcessor.getShowCenterX() - mapProcessor.getCutMapWidth() / 2 <
        2) {
            leftButton.setEnabled(false);
        }
    }
    SwingUtilities.updateComponentTreeUI(smallMapLabel); // 刷新鹰眼漫游区
}
private void rightButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 刷新地图显示区
    getBigMapLabel().setIcon(mapProcessor.moveOfHorizontal(100));
    if (!leftButton.isEnabled()) {
        leftButton.setEnabled(true);
    } else {
        if (mapProcessor.getMap().getWidth() - mapProcessor.getShowCenterX() -
```





```
mapProcessor.getCutMapWidth() / 2 < 2) {  
    rightButton.setEnabled(false);  
}  
}  
SwingUtilities.updateComponentTreeUI(smallMapLabel); // 刷新鹰眼漫游区  
}
```

## 8.6 标记维护

### 8.6.1 功能概述

标记维护是通过弹出菜单实现的，菜单项中提供了创建标记、修改标记、删除标记和查看信息功能，其运行效果如图 8.21 和 8.22 所示。

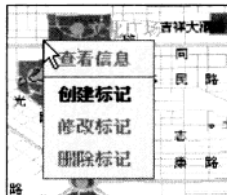


图 8.21 在标记点之外单击鼠标右键



图 8.22 在标记点之上单击鼠标右键

### 8.6.2 创建弹出菜单

在弹出菜单中共有 4 个菜单项，从上到下依次用来查看标记的信息、创建标记、修改标记信息和删除标记，下面是定义这 4 个菜单项的具体代码。

```
// 定义“查看信息”菜单项  
showMenuItem.setText("查看信息"); // 设置菜单项文本  
// 添加事件监听器  
showMenuItem.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        showMenuItemActionPerformed(evt);  
    }  
});  
signPopupMenu.add(showMenuItem); // 添加菜单项到弹出菜单  
signPopupMenu.add(separator); // 添加分隔线  
// 定义“创建标记”菜单项  
createMenuItem.setText("创建标记"); // 设置菜单项文本  
createMenuItem.addActionListener(new java.awt.event.ActionListener() {  
    // 添加事件监听器  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        createMenuItemActionPerformed(evt);  
    }  
});  
signPopupMenu.add(createMenuItem); // 添加菜单项到弹出菜单
```







```
// 定义“修改标记”菜单项
updateMenuItem.setText("修改标记"); // 设置菜单项文本
updateMenuItem.addActionListener(new java.awt.event.ActionListener() {
    // 添加事件监听器
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        updateMenuItemActionPerformed(evt);
    }
});
signPopupMenu.add(updateMenuItem); // 添加菜单项到弹出菜单
// 定义“删除标记”菜单项
deleteMenuItem.setText("删除标记"); // 设置菜单项文本
deleteMenuItem.addActionListener(new java.awt.event.ActionListener() {
    // 添加事件监听器
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        deleteMenuItemActionPerformed(evt);
    }
});
signPopupMenu.add(deleteMenuItem); // 添加菜单项到弹出菜单
```

位于地图显示区的大地图由 **BigMapLabel** 类负责绘制,因此需要为其增加鼠标事件监听。这里仅处理鼠标单击事件,关键代码如下:

```
// 添加鼠标事件监听器
bigMapLabel.addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    // 处理单击鼠标事件
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        bigMapLabelMouseClicked(evt);
    }
});
```

**bigMapLabelMouseClicked()**方法负责处理鼠标单击事件,只有当单击的为鼠标右键时,才显示弹出菜单。在显示之前还要根据单击的位置,判断菜单项的可用性,其关键代码如下:

```
private void bigMapLabelMouseClicked(java.awt.event.MouseEvent evt) {
    if (evt.getButton() == MouseEvent.BUTTON3) { // 查看是否是由单击鼠标右键触发
        boolean isEnteredSign = true; // 默认为在标记之上
        if (getCursor() == Cursor.getDefaultCursor()) {
            isEnteredSign = false; // 未在标记之上
        }
        // 设置菜单项的可用性
        showMenuItem.setEnabled(isEnteredSign);
        createMenuItem.setEnabled(!isEnteredSign);
        updateMenuItem.setEnabled(isEnteredSign);
        deleteMenuItem.setEnabled(isEnteredSign);
        signPopupMenu.show(bigMapLabel, evt.getX(), evt.getY()); // 显示弹出菜单
    }
}
```



**说明** 通过 **MouseEvent** 类的 **getX()**和 **getY()**方法,可以获得鼠标单击位置的横纵

坐标。



237





### 8.6.3 创建和修改标记

创建标记和修改标记对话框有两点不同：一是对话框标题不同，二是文本框中信息不同，其具体效果如图 8.23 和 8.24 所示。

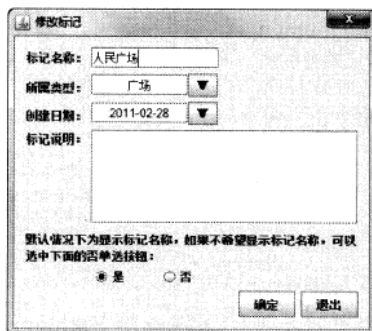
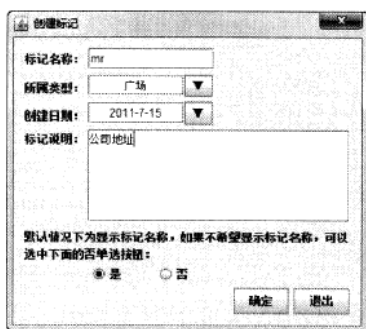


图 8.23 “创建标记”对话框（文本框内容为空） 图 8.24 “修改标记”对话框（文本框内容非空）

单击弹出菜单中的“创建标记”菜单项，将触发 `createMenuItemActionPerformed()` 方法，在该方法中首先判断当前是否存在标记类别，如果存在则创建并显示“创建标记”对话框，其关键代码如下：

```
private void createMenuItemActionPerformed(java.awt.event.ActionEvent evt) {  
    if (dao.selectChildSortByFatherId(0).size() == 0) { // 尚未添加类别  
        JOptionPane.showMessageDialog(this, "请先添加标记类别！", "友情提示",  
            // 弹出提示框  
            JOptionPane.INFORMATION_MESSAGE, null);  
    } else { // 已经添加类别  
        // 显示“创建标记”对话框  
        new ManageSignDialog(null, true, "创建标记").setVisible(true);  
    }  
}
```

单击弹出菜单中的“修改标记”菜单项，将触发 `updateMenuItemActionPerformed()` 方法，在该方法中立即创建并显示“修改标记”对话框，其关键代码如下：

```
private void updateMenuItemActionPerformed(java.awt.event.ActionEvent evt) {  
    // 显示“修改标记”对话框  
    new ManageSignDialog(null, true, "修改标记").setVisible(true);  
}
```

如果创建的是“修改标记”对话框，则需要修改对话框名称，并将原有信息添加到对话框相应位置，其关键代码如下：

```
public ManageSignDialog(java.awt.Frame parent, boolean modal, String title) {  
    super(parent, modal);  
    initComponents();  
    if (title.equals("修改标记")) {  
        setTitle("修改标记"); // 设置对话框名称  
        Vector sign = dao.selectClickSignV(mapProcessor.getRightClickToMapX(),  
            // 获得欲修改标记的信息  
            mapProcessor.getRightClickToMapY());  
        titleTextField.setText(sign.get(2).toString()); // 填写标记名称  
        // 保存所属类型的主键  
        sortTreePanel.getTextField().setName(sign.get(3).toString());  
        // 填写所属类型  
    }  
}
```







```

        sortTreePanel.getTextField().setText(sign.get(4).toString());
        // 填写创建日期
        calendarPanel.getTextField().setText(sign.get(7).toString());
        // 填写标记说明
        remarkTextArea.setText(sign.get(8).toString());
        if ((Integer) sign.get(5) == 0) { // 不显示标记文本
            noRadioButton.setSelected(true);
        }
    }
    ScreenSize.centered(this);
}

```



说明 在默认情况下对话框的名称为“创建标记”。

填写或修改标记信息后单击“确定”按钮，将触发 `submitButtonActionPerformed()` 方法，该方法中首先验证填写信息是否为空，如果为空则弹出提示框，并让该控件获得焦点。否则保存信息到数据库，并刷新地图显示区域。该方法的关键代码如下：

```

private void submitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String title = titleTextField.getText().trim(); // 获得标记名称
    String sortId = sortTreePanel.getTextField().getName(); // 获得所属类型的主键
    String date = calendarPanel.getTextField().getText(); // 获得创建日期
    if (title.length() == 0) { // 标记名称为空
        JOptionPane.showMessageDialog(null, "标记名称不允许为空！", "友情提示",
            JOptionPane.INFORMATION_MESSAGE, null);
        titleTextField.requestFocus(); // 请求获得焦点
        return;
    }
    if (sortId == null) { // 所属类型为空
        JOptionPane.showMessageDialog(null, "所属类型不允许为空！", "友情提示",
            JOptionPane.INFORMATION_MESSAGE, null);
        sortTreePanel.getTextField().requestFocus(); // 请求获得焦点
        return;
    }
    if (date.length() == 0) { // 创建日期为空
        JOptionPane.showMessageDialog(null, "创建日期不允许为空！", "友情提示",
            JOptionPane.INFORMATION_MESSAGE, null);
        calendarPanel.getTextField().requestFocus(); // 请求获得焦点
        return;
    }
    String remark = remarkTextArea.getText().trim(); // 获得标记说明
    int show = 1; // 默认为显示标记文本
    if (noRadioButton.isSelected()) {
        show = 0; // 不显示标记文本
    }
    int x = mapProcessor.getRightClickX(); // 在地图显示区中的水平坐标
    int y = mapProcessor.getRightClickY(); // 在地图显示区中的垂直坐标
    int mapX = mapProcessor.getRightClickToMapX(); // 在地图中的水平坐标
    int mapY = mapProcessor.getRightClickToMapY(); // 在地图中的垂直坐标
    if (getTitle().equals("创建标记")) { // 创建标记
        // 保存到数据库
        int id = dao.insertSign(sortId, mapX, mapY, title, show, mapProcessor
            .getScale(), date, remark);
        // 添加标记
        BigMapLabel.addSign(id, x, y, title, (show == 1 ? true : false));
        // 刷新地图显示区
        SwingUtilities.updateComponentTreeUI(MapPanel.getBigMapLabel());
    } else { // 修改标记
        // 修改标记信息
        BigMapLabel.updateSign(x, y, title, (show == 1 ? true : false));
        // 刷新地图显示区
        SwingUtilities.updateComponentTreeUI(MapPanel.getBigMapLabel());
        // 保存到数据库
    }
}

```





```
        dao.updateSign(mapX, mapY, title, sortId, show, date, remark);
    }
    this.dispose();
}
```

### 8.6.4 删除标记

单击弹出菜单中的“删除标记”菜单项，将触发 `deleteMenuItemActionPerformed()` 方法，在该方法中首先弹出一个用来确认删除的提示框，效果如图 8.25 所示，避免因误操作而删除信息。如果用户单击了“是”按钮，则表示确定要删除选中的标记，该方法的关键代码如下：

```
private void deleteMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    int i = JOptionPane.showConfirmDialog(this, "确定要删除该标记?", "友情提示",
        JOptionPane.YES_NO_OPTION); // 弹出确认删除提示框
    if (i == 0) {
        bigMapLabel.removeSign(); // 确认删除
        SwingUtilities.updateComponentTreeUI(bigMapLabel); // 删除标记
        // 刷新地图显示区
        dao.deleteClickSign(mapProcessor.getRightClickToMapX(),
        mapProcessor.getRightClickToMapY());
        // 从数据库中删除
    }
}
```

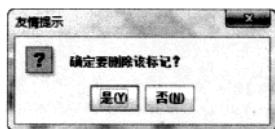


图 8.25 确认删除提示框

### 8.6.5 查看标记信息

如图 8.26 所示的对话框用来查看指定标记的详细信息，对话框中的所有文本框和文本区域均不允许获得焦点，所以“退出”按钮将直接获得焦点，在这种情况下按空格键就可以退出该对话框。

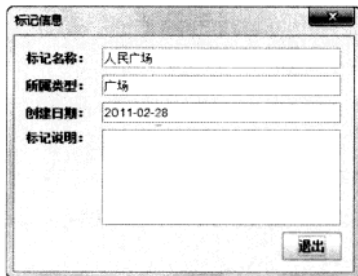


图 8.26 “标记信息”对话框







单击弹出菜单中的“查看信息”菜单项,将触发 `showMenuItemActionPerformed()` 方法,在该方法中立即创建并显示“查看信息”对话框,其关键代码如下:

```
private void showMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    new ShowSignDialog(null, true).setVisible(true); // 显示“查看信息”对话框
}
```

下面是定义对话框中的标签、文本框、文本区域和 `c` 按钮的代码,其中将文本框和文本区域均设置为不允许获得焦点,这样其后的按钮控件将获得焦点。

```
titleLabel.setText("标记名称:");
titleTextField.setFocusable(false); // “标记名称”文本框不允许获得焦点
sortLabel.setText("所属类型:");
sortTextField.setFocusable(false); // “所属类型”文本框不允许获得焦点
dateLabel.setText("创建日期:");
dateTextField.setFocusable(false); // “创建日期”文本框不允许获得焦点
remarkLabel.setText("标记说明:");
remarkTextArea.setColumns(20);
remarkTextArea.setLineWrap(true); // 文本自动折行
remarkTextArea.setRows(5);
remarkTextArea.setFocusable(false); // “标记说明”文本区域不允许获得焦点
remarkScrollPane.setViewportView(remarkTextArea);
exitButton.setText("退出");
exitButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        exitButtonActionPerformed(evt);
    }
});
```



说明 通过执行 Swing 控件对象的 `setFocusable()` 方法,可以不允许控件获得焦点。

单击“退出”按钮,或者在“退出”按钮拥有焦点时按空格键,都会触发“退出”按事件。`exitButtonActionPerformed()` 方法负责处理该事件,它只是销毁了“查看信息”对话框,该方法的关键代码如下:

```
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}
```

## 8.7 标记搜索

### 8.7.1 功能概述

为了方便使用,提供了对标记的搜索功能。搜索分为常用搜索和高级搜索,其运行效果如图 8.27 和 8.28 所示。



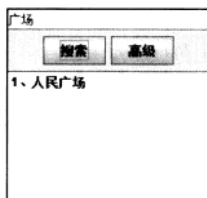


图 8.27 常用搜索

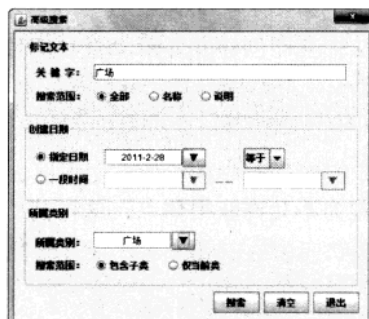


图 8.28 “高级搜索”对话框

## 8.7.2 常用搜索

常用搜索只能以标记文本为条件，同时在标记名称和标记说明中搜索。所有满足条件的标记将显示在搜索结果列表中，其运行效果如图 8.27 所示。

输入搜索关键字后单击“搜索”按钮，将执行 `simpleButtonActionPerformed()` 方法。在该方法中首先判断是否输入了搜索关键字，如果未输入则弹出要求输入关键字的提示框；如果已输入则从数据库中查询符合条件的标记，如果不存在符合条件的标记则弹出提示框，如果存在则设置到列表的模型当中，该方法的关键代码如下：

```
private void simpleButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String general = keywordTextField.getText().trim(); // 获得搜索关键字
    if (general.length() == 0) { // 为输入关键字
        JOptionPane.showMessageDialog(null, "请输入搜索关键字!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE); // 弹出提示框
        return;
    }
    // 查询所有符合条件的标记
    searchResult = dao.selectSimpleSign(general.replace(' ', '%'));
    if (searchResult.size() == 0) { // 不存在符合查询条件的标记
        JOptionPane.showMessageDialog(null, "没有符合条件的标记!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE); // 弹出提示框
    } else { // 存在符合查询条件的标记
        final String[] items = new String[searchResult.size()]; // 创建一个数组对象
        for (int i = 0; i < items.length; i++) { // 遍历符合条件的标记
            // 初始化数组
            items[i] = " " + (i + 1) + "、" + searchResult.get(i).get(4);
        }
        // 为列表框设置模型
        searchResultList.setModel(new javax.swing.AbstractListModel() {
            private static final long serialVersionUID =
                -8613828118413639532L;
            @Override
            public int getSize() { // 返回拥有选项的个数
                return items.length;
            }
            @Override
            public Object getElementAt(int i) { // 返回指定选项
                return items[i];
            }
        });
    }
}
```







### 8.7.3 高级搜索

高级搜索不仅可以使使用标记文本为条件, 还可以将创建日期和标记类别作为搜索条件, 满足搜索条件的标记将显示在搜索结果列表中, 如图 8.28 所示。

单击搜索模块中的“高级”按钮后将执行 `advancedButtonActionPerformed()` 方法, 在该方法中只是创建了一个“高级搜索”对话框, 并令该对话框可见, 其关键代码如下:

```
private void advancedButtonActionPerformed(java.awt.event.ActionEvent evt) {
    new AdvancedSearchDialog(null, true, searchResultList,
        searchResult).setVisible(true);
}
```

确定高级搜索的条件后单击“搜索”按钮, 将执行搜索, 并将搜索结果显示在搜索结果列表中。`searchButtonActionPerformed()` 方法负责处理“搜索”按钮事件, 首先确定标记文本的搜索关键字和搜索范围, 以及所属类别和搜索范围; 然后根据创建日期的搜索条件执行搜索, 该方法的关键代码如下:

```
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String keyword = keywordTextField.getText().trim(); // 获得搜索关键字
    String keywordArea = ""; // 关键字的搜索范围
    Enumeration<AbstractButton> elements = textButtonGroup.getElements();
    while (elements.hasMoreElements()) {
        JRadioButton radioButton = (JRadioButton) elements.nextElement();
        if (radioButton.isSelected()) {
            keywordArea = radioButton.getText();
            break;
        }
    }
    JTextField sortTextField = sortTreePanel.getTextField(); // 获得类别文本框对象
    String sort = (sortTextField.getText().length() > 0 ? sortTextField
        .getName() : null); // 获得搜索类别
    String sortArea = ""; // 类别的搜索范围
    if (childTreeRadioButton.isSelected()) {
        sortArea = "包含子类";
    } else {
        sortArea = "仅当前类";
    }
    if (appointRadioButton.isSelected()) { // 指定日期
        String date = appointPanel.getTextField().getText(); // 获得日期
        // 获得比较条件
        String compare = compareComboBox.getSelectedItem().toString();
        if (keyword.length() > 0 || sort != null || date.length() > 0) {
            // 输入了查询条件
            dealWithSearchResult(dao.selectAppointSign(keyword, keywordArea,
                date, compare, sort, sortArea));
        } else { // 未输入任何查询条件
            JOptionPane.showMessageDialog(null, new String[] { "至少填写: ", "
                标记文本",
                "创建日期", " 所属类别", "中的一个搜索条件!" },
                "友情提示", JOptionPane.INFORMATION_MESSAGE);
        }
    } else { // 一段时间
        String startDate = spaceStartPanel.getTextField().getText(); // 开始日期
        String endDate = spaceEndPanel.getTextField().getText(); // 结束日期
        int s1 = startDate.length();
        int e1 = endDate.length();
    }
}
```







```
if (s1 * e1 != 0) { // 两个日期全部选择了
    DateFormat df = DateFormat.getDateInstance();
    Date sd = null;
    Date ed = null;
    try {
        sd = df.parse(startDate);
        ed = df.parse(endDate);
    } catch (ParseException ex) {
        Logger.getLogger(AdvancedSearchDialog.class.getName()).
log(Level.SEVERE, null, ex);
    }
    if (ed.before(sd)) {
        JOptionPane.showMessageDialog(null, "“终止日期”不能在“起始日期”
之前, 请重新选择!", "友情提示", JOptionPane.INFORMATION_MESSAGE);
    } else {
        dealWithSearchResult(dao.selectSpaceSign(keyword, keywordArea,
startDate, endDate, sort, sortArea));
    }
} else { // 存在未选择的日期
    if (s1 + e1 != 0) { // 只选择了一个日期
        if (s1 == 0) { // 未选择开始日期
            JOptionPane.showMessageDialog(null, "请填写“起始日期”!",
"友情提示",
JOptionPane.INFORMATION_MESSAGE);
        } else { // 未选择结束日期
            JOptionPane.showMessageDialog(null, "请填写“终止日期”!",
"友情提示",
JOptionPane.INFORMATION_MESSAGE);
        }
    } else { // 未选择任何日期
        if (keyword.length() > 0 || sort != null) { // 输入了查询条件
            dealWithSearchResult(dao.selectAppointSign(keyword,
keywordArea, "", "", sort, sortArea));
        } else { // 未输入任何查询条件
            JOptionPane.showMessageDialog(null, new String[] { "至少填写:",
"标记文本", "创建日期", "所属类别",
"中的一个搜索条件!" },
"友情提示", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
}
```



#### 8.7.4 描红并居中显示标记



单击“搜索结果”列表中的某一标记后, 该标记将显示在地图显示区的中央, 并且为描红显示, 具体效果如图 8.29 所示。





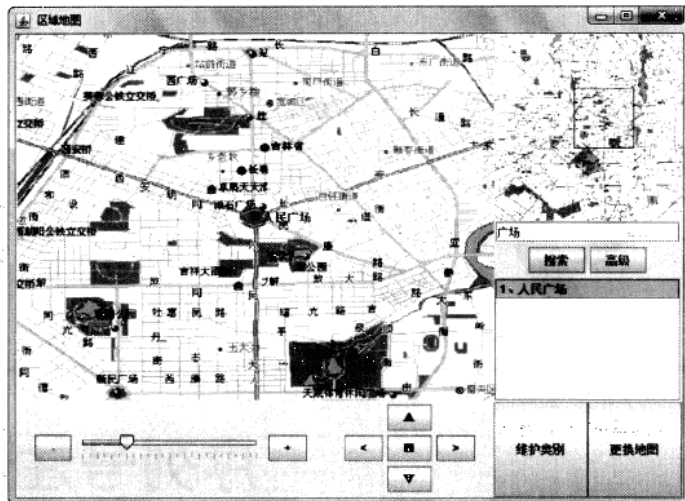


图 8.29 描红并居中显示选中标记

`searchResultListValueChanged()`方法负责处理列表框选项选中事件。首先利用选中项的索引获得相应标记的信息,其次调整地图处理器中的视觉中心,再次将该标记设置为描红显示,最后根据缩放比例刷新地图,该方法的关键代码如下:

```
private void searchResultListValueChanged(javax.swing
.event.ListSelectionEvent evt) {
    int selectedIndex = searchResultList.getSelectedIndex(); // 获得选中项索引
    Vector sign = searchResult.get(selectedIndex);           // 获得选中项向量
    mapProcessor.adjustShowCenter((Integer) sign.get(2), (Integer)
sign.get(3)); // 调整视觉中心
    bigMapLabel.setStress((Integer) sign.get(0));           // 设置为描红显示
    // 刷新地图
    int sliderValue = slider.getValue();                     // 获得当前的缩放比例
    // 获得需要的缩放比例
    int scaleValue = mapProcessor.parseScale((Double) sign.get(6));
    if (sliderValue < scaleValue) {                          // 当前比例小于缩放比例
        slider.setValue(scaleValue);
    } else {                                                 // 当前比例不小于缩放比例
        slider.setValue(sliderValue + 1);
    }
}
```

# 第 9 章

---

## 序列号注册模块

(Swing+RSA 实现)

程序开发是一项极为耗费时间和精力的工作，对于优秀的软件更是如此。对于开发人员而言，也需要考虑个人生计问题，因此软件注册势在必行。比较常见的注册方式有电话注册、联网注册、Key 文件注册等。本章将开发一个序列号注册模块，读者可以将该模块应用到自己开发的软件中。通过本章的学习，读者可以学到：

- » 使用 Swing 技术设计桌面程序
- » 注册机的实现
- » 验证用户名和注册码





## 9.1 序列号注册模块概述

### 9.1.1 模块概述

序列号注册模块主要用于限制用户使用软件的时间、保证软件仅能在一台电脑上使用，并为软件提供了试用功能。当超过试用期后，如果希望继续试用，需要获取注册码完成软件的合理注册。

除了上述功能外，本模块还为用户提供了注册机，它可以根据用户输入的相关信息生成 20 位注册码，用户可以使用本模块来限制自己开发的软件的使用。

### 9.1.2 功能结构

在本模块中主要包括软件试用、软件注册和注册机 3 个部分，其中软件试用部分主要包括“软件试用时间限制”、“防止用户修改系统时间”两个功能，软件注册部分主要包括“软件注册时间限制”、“将注册信息写入注册表”、“加密注册信息”等功能，而注册机部分主要包括“生成注册码”功能。序列号注册模块的功能结构图如图 9.1 所示。

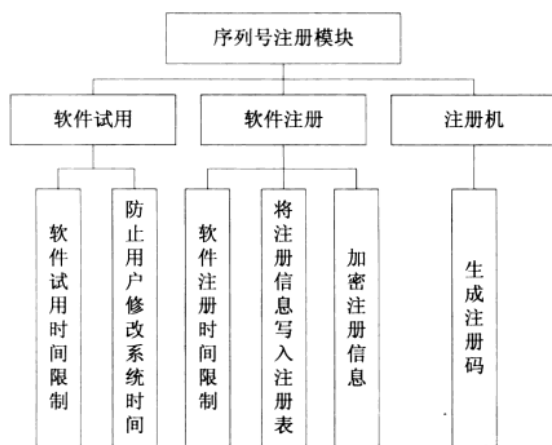


图 9.1 序列号注册模块功能结构图

### 9.1.3 程序预览

序列号注册模块主要由 3 个窗体构成，主要包括注册导航窗体、软件注册窗体及注册机窗体。注册导航窗体主要为用户提供软件注册与软件试用选择功能，其运行效果如图 9.2 所示。

软件注册窗体为用户提供注册功能。用户在使用注册机获取注册码后，需要填写用户





名与注册码。如果正确，则弹出注册成功对话框。其运行效果如图 9.3 所示。

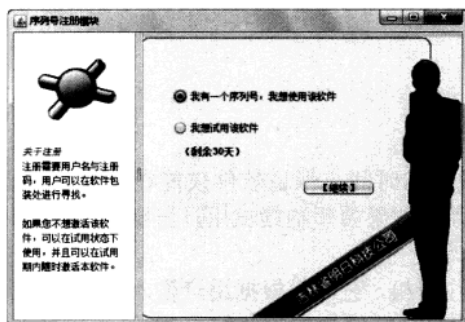


图 9.2 注册导航窗体

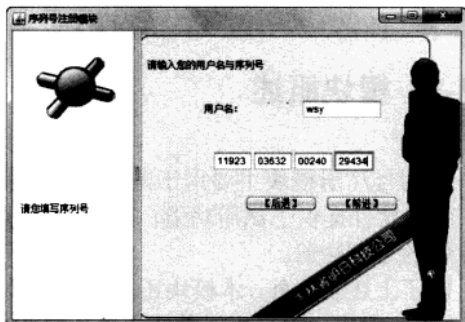


图 9.3 软件注册窗体

注册机窗体主要完成根据用户名生成注册码的功能，其运行效果如图 9.4 所示。

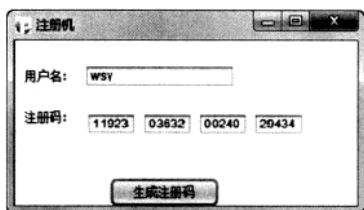


图 9.4 注册机窗体

## 9.2 关键技术

### 9.2.1 读取客户端 MAC 地址

在通常情况下，生成的注册码与客户端计算机硬件信息相关，例如，使用 MAC 地址。获取 MAC 地址需要了解当前的操作系统。对于不同的 Windows 系统，cmd 命令所在位置也不同。首先使用 System 类的 getProperties()方法获得操作系统信息，然后根据 Windows 系统的类型确定 CMD 命令位置。getProperties()方法可以确定当前系统属性，其参数是一些固定的键。常用的键及其表示的意义如表 9.1 所示：

表 9.1 System 类中键与含义对应表

键	含义
java.version	Java 运行时环境版本
java.home	Java 安装目录
os.name	操作系统的名称
os.version	操作系统的版本
user.name	用户的账户名称

在确定 CMD 命令所在位置后，使用 Runtime 类中的 exec()方法来执行指定 DOS 命令，







并将返回值放置在 `InputStream` 对象中, 遍历并获取带有“物理地址”的行信息, 提取该行中的 MAC 地址。获取 MAC 地址的 `getMAC()` 方法关键代码如下:

```
public static String getMAC() {
    String macResult = null;
    String osName = System.getProperty("os.name");    // 获取操作系统的名称
    // 实例化 StringBuffer 对象
    StringBuffer systemPathBuff = new StringBuffer("");
    if (osName.indexOf("Windows") > -1) {
        // Windows 操作系统的 cmd.exe 的绝对路径
        systemPathBuff.append("c:\\WINDOWS\\system32\\cmd.exe");
    }
    if (osName.indexOf("NT") > -1) {
        // NT 操作系统 cmd.exe 的绝对路径
        systemPathBuff.append("c:\\WINDOWS\\command.exe");
    }
    Process pro = null;
    try {
        // 此语句相当于在 cmd 下面执行 dir 命令, 并得到命令执行完毕后的输出流
        // 执行 DOS 命令
        pro = Runtime.getRuntime().exec(systemPathBuff.toString() + " /c dir");
        InputStream is = pro.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        String message = br.readLine();    // 读取第一行
        int index = -1;
        message = br.readLine();    // 读取下一行
        // 此语句相当于在 cmd 下面执行 ipconfig/all 的命令, 并得到命令执行完毕后的输出流
        pro = Runtime.getRuntime().exec(systemPathBuff.toString() + " /c ipconfig/all");
        is = pro.getInputStream();
        br = new BufferedReader(new InputStreamReader(is));
        message = br.readLine();    // 读取第一行
        while (message != null) {
            if ((index = message.indexOf("物理地址")) > 0) { // 找到 MAC 地址那行
                // 将结果放置在 macResult 变量中
                macResult = message.substring(index + 32).trim();
                break;
            }
            message = br.readLine();    // 读取下一行
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return macResult;    // 将 MAC 地址返回
}
```



对于 Windows 7 操作系统而言, 需要使用“物理地址”来查找 MAC 值。如果是 Windows XP 系统, 可能需要使用“Physical Address”查找 MAC 地址。



249



## 9.2.2 Java 操作注册表

操作注册表即对注册表进行读写操作。在 JDK1.4 版之后, 可以使用 `java.util.prefs` 包中的 `Preferences` 类实现这一功能, 使用该类操作注册表的步骤如下。



(1) 获得 Preferences 类的对象。由于该类的构造方法被 protected 关键字修饰, 因此不能直接使用 new 关键字进行实例化, 通常使用静态方法 userRoot()和 systemRoot()来完成实例化, 这两个方法的声明如下:

```
public static Preferences userRoot()
public static Preferences systemRoot()
```

(2) 使用 node()方法将指定首选项阶段返回到此节点所在的同一树中, 该方法的声明如下:

```
public abstract Preferences node(String pathName)
pathName: 要返回的首选项节点的路径名。
```

(3) 使用 put()、putInt()或 putDouble()方法为注册表中的相关项赋值。

笔者将读、写注册表内容代码分别封装在两个方法中, 名称分别为 readValue()和 writeValue(), 它们的关键代码如下:

```
public static String readValue(String node, String key) {
    Preferences pre = Preferences.systemRoot().node(node);
    return pre.get(key, "注册表中没有该值");
}
public static void writeValue(String node, String[] keys, Object[] values) {
    Preferences pre = Preferences.systemRoot().node(node);
    for (int i = 0; i < keys.length; i++) {
        pre.put(keys[i], String.valueOf(values[i]));
    }
}
public static void writeValue(String node, String keys, String values) {
    Preferences pre = Preferences.systemRoot().node(node);
    pre.put(keys, values);
}
```

### 9.2.3 避免用户修改系统时间

在软件使用到期后, 用户可以修改系统时间以便继续使用, 为了避免这种情况, 需要对此进行限制。当用户驱动软件注册导航窗体时, 程序将相关信息写入注册表。它们包括用户第一次使用时间 programe1、系统当前时间 programe2 及用户使用该软件状态 programe3。如果用户修改系统时间, 当前时间一定会小于 programe2 中记录的时间, 此时将弹出错误提示对话框, 并终止使用。其关键代码如下:

```
if (ReadWriteRegisty.readValue("myprograme", "programe1").equals("注册表中没有该值")) {
    // ZHUCE=-1 代表试用时间满, ZHUCE=-2 代表已注册, 其余数字代表该软件使用的天数
    // 节点第一个字母不要设置为大写, 否则在注册表中会添加一个"\ "字样
    String[] key = { "programe1", "programe2", "programe3" };
    String d = dateFormate.format(new Date());
    Object[] value = { d, d, "sign1" };
    // 注册表中 programe1 键值代表首次运行本软件的时间
    // programe2 键值代表当前运行软件时间
    // programe3 代表当前软件状态, sign1 代表软件在试用期中, sign2 代表软件试用期满
    // sign3 代表已经注册, sign4 代表注册期满
    ReadWriteRegisty.writeValue("myprograme", key, value);
} else {
    try {
        // 如果修改系统日期, 希望继续使用该软件, 比较两个时间的先后
```







```

        if (new Date().compareTo(df.parse(ReadWriteRegistry.readValue(
            "myprogram", "programe2")))) <= 0) {
            JOptionPane.showMessageDialog(mainFrame.this,
                "软件已经试用到期, 如果您想继续使用, 请购买序列号进行注册使用");
            ReadWriteRegistry.writeValue("myprogram", "programe3", "sign2");
            ZHUCE = -1;
        }
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
if (ZHUCE > 0 || ReadWriteRegistry.readValue("myprogram", "programe3").
    equals("sign1")) {
    try {
        ReadWriteRegistry.writeValue("myprogram", "programe2", dateFormat.
            format(new Date()));
        SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date_start=df2.parse(new Date().toLocaleString());
        Date date_end = df2.parse(ReadWriteRegistry.readValue("myprogram",
            "programe1"));
        Calendar cal_start = Calendar.getInstance();
        cal_start.setTime(date_start);
        Calendar cal_end = Calendar.getInstance();
        cal_end.setTime(date_end);
        ZHUCE = days - getDifferenceDays(cal_start, cal_end);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
}
}

```

### 9.2.4 弹出菜单

在软件注册窗体中, 有4个用于接收注册码的文本框, 如果用户在文本框中单击鼠标右键, 将弹出菜单, 其运行效果如图9.5所示。

实现该功能的步骤如下。

(1) 为文本框添加鼠标单击事件监听。由于这几个文本框实现的事件相同, 因此将其封装到一个类中, 从而达到了代码重用。

(2) 在监听器中判断用户是否单击鼠标右键, 从而决定是否初始化弹出式菜单。

(3) 为弹出式菜单增加菜单项。

(4) 为菜单项增加事件监听器, 实现剪切、复制、粘贴功能。

其关键代码如下:

```

class JCopy extends MouseAdapter {
    @Override
    public void mouseDragged(MouseEvent arg0) {
        super.mouseDragged(arg0);
    }
}

```

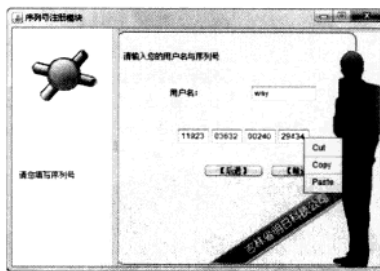


图 9.5 软件注册窗体







```
@Override
public void mouseClicked(MouseEvent arg0) {
    super.mouseClicked(arg0);
    // 鼠标按键 getButton() 方法返回 1 表示按了鼠标左键, 2 表示按了鼠标中键, 3 表示按了
鼠标右键
    if (arg0.getButton() == 3) {
        // 如果用户单击鼠标右键
        final JPopupMenu popup = new JPopupMenu(); // 初始化弹出式菜单
        JMenuItem itemcut = new JMenuItem("Cut"); // 初始化剪切菜单项
        popup.add(itemcut); // 将菜单项添加到菜单中
        popup.addSeparator(); // 添加分割线
        JMenuItem itemcopy = new JMenuItem("Copy"); // 初始化复制菜单项
        popup.add(itemcopy); // 将菜单项添加到菜单中
        // 为复制菜单项添加事件监听器
        itemcopy.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                t1.selectAll();
                t2.selectAll();
                t3.selectAll();
                t4.selectAll();
                JTextJP.requestFocus(true);
                System.out.println(JTextJP.requestFocusInWindow());
                String t1Selection = t1.getSelectedText();
                String t2Selection = t2.getSelectedText();
                String t3Selection = t3.getSelectedText();
                String t4Selection = t4.getSelectedText();
                popup.setVisible(false);
                String IntegrationString = t1Selection.concat("-").concat(
                    t2Selection).concat("-").concat(t3Selection).
                    concat("-").concat(t4Selection);
                if (IntegrationString == null)
                    return;
                StringSelection clipString = new StringSelection
                    (IntegrationString);
                clipbd.setContents(clipString, clipString);
            }
        });
        popup.addSeparator();
        JMenuItem itempaste = new JMenuItem("Paste");
        popup.add(itempaste);
        itempaste.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                Transferable clipData = clipbd.getContents(RegisterFrame.
                    this);
                try {
                    String clipString = (String) clipData.getTransferData
                        (DataFlavor.stringFlavor);
                    if (clipString.contains("-")) {
                        String[] a = clipString.split("-");
                        for (int i = 0; i < a.length; i++)
                            System.out.println(a[i]);
                        t1.setText(a[0]);
                        t2.setText(a[1]);
                        t3.setText(a[2]);
                        t4.setText(a[3]);
                    } else {
                        JOptionPane.showMessageDialog(RegisterFrame.this,
                            "您粘贴的注册码格式不正确, 请重新粘贴");
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```







```

        }
        popup.setVisible(false);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
));
popup.show(arg0.getComponent(), arg0.getX(), arg0.getY());
}
}
}

```

### 9.2.5 一次性粘贴注册码

为了方便使用,在填写注册码时,提供了一次性粘贴功能,即当用户复制一串带有“-”分隔符的字符串时,可以一次性粘贴到软件注册窗体的4个文本框中。为了实现这一功能,需要了解Java中剪切板技术。

在图形界面程序中,复制和粘贴是最有用的用户接口之一。可以使用剪切板将文字、图像从一个文档移动到另一个文档中,实现剪切板功能要用到java.awt.datatransfer包。实现该功能的关键代码如下:

```

String clipString = (String) clipData.getTransferData(DataFlavor.
stringFlavor);
if (clipString.contains("-")) {
    String[] a = clipString.split("-");
    for (int i = 0; i < a.length; i++)
        System.out.println(a[i]);
    t1.setText(a[0]);
    t2.setText(a[1]);
    t3.setText(a[2]);
    t4.setText(a[3]);
} else {
    JOptionPane.showMessageDialog(RegisterFrame.this, "您粘贴的注册码格式不正确,
请重新粘贴");
}
}

```

### 9.2.6 计算两个时间的间隔天数

在序列号注册模块实现过程中,有多处需要比较两个时间,如避免用户修改系统时间、判断用户试用时间、验证用户注册时间等。

Java中获取两个时间的差有两种方式:第一种是使用getTime()方法来获得两个时间的毫秒差,然后转化成天数;第二种是使用Calendar类计算两个时间间隔天数。本模块使用第二种,其关键代码如下:

```

SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
Date date_start=df2.parse(new Date().toLocaleString());
Date date_end = df2.parse(ReadWriteRegisty.readValue("myprograme",
"program1"));
Calendar cal_start = Calendar.getInstance();
cal_start.setTime(date_start);
Calendar cal_end = Calendar.getInstance();

```



253





```
cal_end.setTime(date_end);
ZHUCE = days - getDifferenceDays(cal_start, cal_end);
```

在上述代码中使用自定义的 `getDifferenceDays()` 方法计算两个 `Calendar` 对象间隔天数。在 `Calendar` 类中有多个成员变量，其中 `DAY_OF_YEAR` 可以获得当前对象在一年中的天数。`getDifferenceDays()` 方法关键代码如下：

```
public static int getDifferenceDays(Calendar d1, Calendar d2) {
    if (d1.after(d2)) {
        java.util.Calendar swap = d1;
        d1 = d2;
        d2 = swap;
    }
    int days = d2.get(java.util.Calendar.DAY_OF_YEAR) - d1.get(java.
        util.Calendar.DAY_OF_YEAR);
    int y2 = d2.get(java.util.Calendar.YEAR);
    if (d1.get(java.util.Calendar.YEAR) != y2) {
        do {
            days += d1.getActualMaximum(java.util.Calendar.DAY_OF_YEAR);
            d1.add(java.util.Calendar.YEAR, 1);
        } while (d1.get(java.util.Calendar.YEAR) != y2);
    }
    return days;
}
```

上述代码获取了两个时间相差天数。在计算这个天数之前，需要将大的日期放置在小日期前面，然后进行相减操作，上述代码除了可以计算在同一年中的两个时间的相差天数之外，还可以进行跨年计算，判断两个日期是否在同一年，如果不在，将前面的日期反复加 1，直到与后面日期的年数相等为止，同时记得要把这一年的天数加到两个日期相差的天数上。

## 9.2.7 ini 文件的读写

ini 类型的文件用于存储配置信息，笔者使用它来保存注册用户硬件相关信息，下面介绍如何读写该文件。

`Properties` 类位于 `java.util` 包中，它可以辅助 ini 类型文件读写，其使用步骤如下：

- (1) 使用 `put()` 方法将键与值进行对应，然后使用 `store()` 方法将该对象存入文件中。
- (2) 使用 `load()` 方法加载 ini 文件，然后使用 `get()` 方法获取相应字符串的值。

本模块中，使用 `Properties` 类写入 ini 文件的关键代码如下：

```
Properties pro = new Properties();
pro.put("register", String.valueOf(codeStringBigint));
try {
    // 将信息保存在 a.ini 文件中
    pro.store(new FileOutputStream("a.ini", true), null);
    // 可以从 a.ini 中通过 Properties.get 方法读取配置信息
    pro.load(new FileInputStream("a.ini"));
} catch (Exception ex) {
    ex.printStackTrace();
}
```

使用 `Properties` 类读取 ini 文件的关键代码如下：

```
Properties pro = new Properties();
```







```
try {
    // 将信息保存在 a.ini 文件中
    pro.store(new FileOutputStream("a.ini", true), null);
    // 可以从 a.ini 中通过 Properties.get 方法读取配置信息
    pro.load(new FileInputStream("a.ini"));
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
String ctext = String.valueOf(String.valueOf(pro.get("register")));
BigInteger c = new BigInteger(ctext);
```

执行上述代码后, 当前目录下会出现一个名为 a.ini 的文件, 打开该文件, 运行效果如图 9.6 所示。

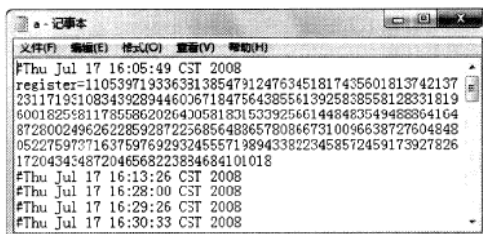


图 9.6 a.ini 文件

## 9.2.8 RSA 加密/解密算法

在本模块中, 考虑到安全因素, 笔者将安装软件的硬件信息加密, 然后写入配置文件中, 这里使用 RSA 加密/解密算法。

RSA 算法是常用的非对称加密算法, 它的基本原理就是基于大数的因子分解, 将明文看做是一个无符号的大整数。RSA 的密钥长度为 40~2048 位可变, 加密时将明文分成块, 块的大小可以改变, 但每个块的长度不能超过密钥的长度, 这就需要传输的数据分段传输, 密钥的长度越长保密性越好, 但密钥的长度越长加密与解密的效率越低。

使用 RSA 加密解密的步骤如下。

(1) 生成 RSA 密钥对, 使用 java.security 包中的一些相关类获取私钥与共钥, 然后将公钥与私钥写入相应文件中, 其关键代码如下:

```
public genKey() {
    try {
        // 创建密钥对生成器
        KeyPairGenerator KPG = KeyPairGenerator.getInstance("RSA");
        KPG.initialize(1024); // 初始化该生成器
        KeyPair KP = KPG.genKeyPair(); // 生成密钥对
        PublicKey pbkey = KP.getPublic(); // 获取共钥
        PrivateKey prkey = KP.getPrivate(); // 获取私钥
        // 将共钥写进文件中
        FileOutputStream filePbkey = new FileOutputStream("RSAPubkey.dat");
        ObjectOutputStream filePbkeyStream = new ObjectOutputStream(filePbkey);
        filePbkeyStream.writeObject(pbkey);
        // 将私钥写进文件中
```



255







```
        FileOutputStream filePrkey = new FileOutputStream("RSAPrikey.dat");
        ObjectOutputStream filePrkeyStream = new ObjectOutputStream(filePrkey);
        filePrkeyStream.writeObject(prkey);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(2) 根据密钥进行 RSA 加密处理。使用 `java.secrity.RSAPublicKey` 类中的 `getPublicExponent()`方法和 `getModulus()`方法可以得到 RSA 算法公式的两个参数，然后将明文转型为 `BigInteger` 类型。由于 `BigInteger` 类中的 `modPow()`方法可执行 RSA 算法公式，所以将该上述两个参数作为 `modPow()`方法的参数即可完成加密操作，其关键代码如下：

```
try {
    File f = new File("a.ini");
    if (!f.exists()) {
        new genKey();
        // 读取公钥
        FileInputStream FIS = new FileInputStream("RSAPubKey.dat");
        ObjectInputStream OIS = new ObjectInputStream(FIS);
        RSAPublicKey RSAPK = (RSAPublicKey) OIS.readObject();
        // 得到两个公钥的参数
        BigInteger e = RSAPK.getPublicExponent();
        BigInteger n = RSAPK.getModulus();
        // 将明文转为大整数
        byte[] strByte = (str.concat(register)).getBytes("UTF8");
        BigInteger m = new BigInteger(strByte);
        codeStringBigint = m.modPow(e, n); // 进行加密操作
        Properties pro = new Properties(); // 保存密文
        pro.put("register", String.valueOf(codeStringBigint));
        try {
            // 将信息保存在 a.ini 文件中
            pro.store(new FileOutputStream("a.ini", true), null);
            // 可以从 a.ini 中通过 Properties.get 方法读取配置信息
            pro.load(new FileInputStream("a.ini"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

在使用 RSA 对密文进行解密的操作中，原理基本上与加密时相同。需要获取解密密钥的两个参数，然后将密文转型为 `BigInteger` 类型，调用 `modPow()`方法进行解密操作，将解密后的明文转型为 `Byte` 类型的数组，读取数组中的每个值，强制转换为字符形式。其关键代码如下：

```
String ctext = String.valueOf(String.valueOf(pro.get("register")));
BigInteger c = new BigInteger(ctext);
// 获取私钥
FileInputStream f = new FileInputStream("RSAPrikey.dat");
ObjectInputStream b = new ObjectInputStream(f);
RSAPrivateKey prk = (RSAPrivateKey) b.readObject();
// 得到公钥的两个参数
BigInteger d = prk.getPrivateExponent();
BigInteger n = prk.getModulus();
// 解密处理
```







```

BigInteger m = c.modPow(d, n);
byte[] mt = m.toByteArray();
for (int i = 0; i < mt.length; i++) {
    s += (char) mt[i];
}

```

## 9.3 软件注册导航窗体

### 9.3.1 功能概述

软件注册导航窗体主要为用户提供试用与软件注册选择功能,其运行效果如图 9.7 所示。

在软件注册导航窗体中,如果用户选择“我想试用该软件”单选按钮,并单击“继续”按钮时,软件进入试用状态,如果用户选择“我有一个序列号,我想使用该软件”单选按钮,并单击“继续”按钮时,软件进入软件注册窗体。

软件注册窗体的布局主要包括左方的介绍面板及右方的选择面板,在这里使用到了 JSplitPane 控件,设定整个窗体以左右方式进行分割,其关键代码如下:

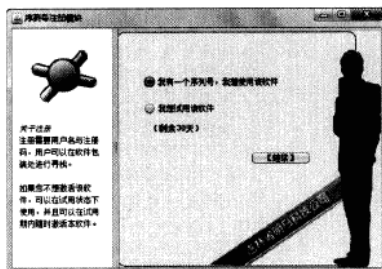


图 9.7 注册导航窗体

```

public mainFrame() {
    super();
    try {
        UIManager.setLookAndFeel(new
com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel());
    } catch (UnsupportedLookAndFeelException e1) {
        e1.printStackTrace();
    }
    ButtonGroup bg = new ButtonGroup();
    final JRadioButton b1, b2;
    JButton b3;
    final JSplitPane splitPane = new JSplitPane();
    splitPane.setDividerLocation(150);
    JPanel rightPanel = new JPanel();
    ImageIcon icon = CreatecdIcon.add("11.jpg");
    JLabel jl = new JLabel();
    jl.setLayout(null);
    jl.setIcon(icon);
    jl.setBounds(0, 0, icon.getIconWidth(), icon.getIconHeight());
    b1 = new JRadioButton("我有一个序列号,我想使用该软件");
    b1.setBackground(new Color(234, 241, 247));
    b1.setSelected(true);
    b2 = new JRadioButton("我想试用该软件");
    jl.add(b1);
    b1.setBounds(40, 60, 260, 30);
    jl.add(b2);
    b2.setBackground(new Color(234, 241, 247));
    b2.setBounds(40, 100, 200, 30);
    bg.add(b1);
    bg.add(b2);
}

```







```
b3 = new JButton("【继续】");
j1.add(b3);
b3.setBounds(200, 180, 90, 20);
j1.add(zhuce);
zhuce.setBounds(50, 130, 200, 30);
int days = 30; //允许试用的天数
final SimpleDateFormat dateFormate = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
final DateFormat df = DateFormat.getDateInstance();
// 判断是否首次使用该软件
if (ReadWriteRegistry.readValue("myprograme", "programe1").equals("注册表中
没有该值")) {
    // ZHUCE=-1 代表试用时间满, ZHUCE=-2 代表已注册, 其余数字代表该软件使用的天数
    // 节点第一个字母不要设置为大写, 否则在注册表中会添加一个“\”字样
    String[] key = { "programe1", "programe2", "programe3" };
    String d = dateFormate.format(new Date());
    Object[] value = { d, d, "sign1" };
    // 注册表中 programe1 键值代表首次运行本软件的时间
    // programe2 键值代表当前运行软件时间
    // programe3 代表当前软件状态, sign1 代表软件在试用期中, sign2 代表软件试用期满
    // sign3 代表已经注册, sign4 代表注册期满。
    ReadWriteRegistry.writeValue("myprograme", key, value);
} else {
    try {
        // 如果修改系统日期, 希望继续使用该软件
        // 比较两个时间的先后
        if (new Date().compareTo(df.parse(ReadWriteRegistry.readValue
("myprograme", "programe2")))) <= 0) {
            JOptionPane.showMessageDialog(mainFrame.this,
                "软件已经试用到期, 如果您想继续使用, 请购买序列号进行注
册使用");
            ReadWriteRegistry.writeValue("myprograme", "programe3", "sign2");
            ZHUCE = -1;
        }
    } catch (ParseException e) {
        e.printStackTrace();
    }
    if (ZHUCE > 0 || ReadWriteRegistry.readValue("myprograme", "programe3").
equals("sign1")) {
        try {
            ReadWriteRegistry.writeValue("myprograme", "programe2",
                dateFormate.format(new Date()));
            SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            Date date_start=df2.parse(new Date().toLocaleString());
            Date date_end = df2.parse(ReadWriteRegistry.readValue
("myprograme", "programe1"));
            Calendar cal_start = Calendar.getInstance();
            cal_start.setTime(date_start);
            Calendar cal_end = Calendar.getInstance();
            cal_end.setTime(date_end);
            ZHUCE = days - getDifferenceDays(cal_start, cal_end);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
if (ZHUCE <= 0 || ReadWriteRegistry.readValue("myprograme", "programe3").
equals("sign2")) {
    ZHUCE = 0;
}
```





```

zhuze.setText("(剩余" + ZHUZE + "天)");
b3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        if (b1.isSelected()) {
            setVisible(false);
            new RegisterFrame().setVisible(true);
        }
        if (b2.isSelected()) {
            if (ReadWriteRegisty.readValue("myprograme", "programe3").
                equals("sign2")) {
                JOptionPane.showMessageDialog(mainFrame.this,
                    "软件已经试用到期,如果您想继续使用,请购买序列号进行注册使用");
                return;
            }
            if (ZHUZE == 0) {
                JOptionPane.showMessageDialog(mainFrame.this,
                    "软件已经试用到期,如果您想继续使用,请购买序列号进行注册使用");
                ReadWriteRegisty.writeValue("myprograme", "programe3", "sign2");
                return;
            }
        }
    }
});
rightPanel.add(j1, new Integer(Integer.MIN_VALUE));
JPanel leftPanel = new JPanel();
leftPanel.setLayout(null);
leftPanel.setBackground(Color.WHITE);
JLabel image = new JLabel();
ImageIcon icon2 = CreatecdIcon.add("00002.jpg");
image.setIcon(icon2);
JLabel letter1 = new JLabel();
letter1.setText("<html><i>关于注册</i><br>注册需要用户名与注册码,用户可以在软件  
包装处进行寻找.<br><br>如果您不想激活该软件,可以在试用状态下使用,并且可以在试用期内随时激活  
本软件.</html>");
JLabel letter2 = new JLabel();
leftPanel.add(image);
image.setBounds(20, 20, 130, 100);
leftPanel.add(letter1);
letter1.setBounds(10, 120, 130, 200);
leftPanel.add(letter2);
letter2.setBounds(10, 200, 130, 50);
splitPane.setLeftComponent(leftPanel);
splitPane.setRightComponent(rightPanel);
getContentPane().add(splitPane);
setSize(new Dimension(570, 400));
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setTitle("序列号注册模块");
}

```

### 9.3.2 软件试用功能实现

在软件试用功能中,首先要将用户试用时间及当前时间和窗体状态写入注册表,避免用户修改系统时间延长试用期限,其实现步骤如下。

(1) 软件启动后首先判断注册表 HKEY\_LOCAL\_MACHINE\Software\MyProgram 子







键下的3个键值中是否有值。如果没有值,说明用户是第一次试用该软件,将这3个键值的初始值写入注册表。其中 program1 键值代表首次运行本软件的时间, program2 代表当前运行软件的时间,而 program3 代表当前软件的使用状态, program3 键值对应的值分别为 sign1、sign2、sign3、sign4,分别表软件在试用期间、软件试用期满、用户已注册、注册期满。

(2) 如果用户已经试用该软件,需要判断用户是否超过试用期限。如果超过期限,将弹出错误提示对话框,并将注册表中试用该软件的状态修改为 sign2。

(3) 更新软件注册导航窗体中的试用剩余天数,这里试用天数放置在一个名为 ZHUCE 的全局变量中,这个变量依据读取注册表中的 program1 键值而赋值, ZHUCE 为 -1 说明试用期满, ZHUCE 为 -2 说明软件已经注册, ZHUCE 为其他数字表示试用剩余天数。在该窗体中将 ZHUCE 变量放置在标签中表示试用天数。

其关键代码如下:

```
int days = 30; // 允许试用的天数
final SimpleDateFormat dateFormate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
final DateFormat df = DateFormat.getDateInstance();
// 判断是否首次使用该软件
if (ReadWriteRegisty.readValue("myprogram", "program1").equals("注册表中没有该值")) {
    // ZHUCE=-1 代表试用时间满, ZHUCE=-2 代表已注册, 其余数字代表该软件使用的天数
    // 节点第一个字母不要设置为大写, 否则在注册表中会添加一个 "\" 字样
    String[] key = { "program1", "program2", "program3" };
    String d = dateFormate.format(new Date());
    Object[] value = { d, d, "sign1" };
    // 注册表中 program1 键值代表首次运行本软件的时间
    // program2 键值代表当前运行软件时间
    // program3 代表当前软件状态, sign1 代表软件在试用期中, sign2 代表软件试用期满
    // sign3 代表已注册, sign4 代表注册期满
    ReadWriteRegisty.writeValue("myprogram", key, value);
} else {
    try {
        // 如果修改系统日期, 希望继续使用该软件
        // 比较两个时间的先后
        if (new Date().compareTo(df.parse(ReadWriteRegisty.readValue("myprogram", "program2"))) <= 0) {
            JOptionPane.showMessageDialog(mainFrame.this,
                "软件已经试用到期, 如果您想继续使用, 请购买序列号进行注册使用");
            ReadWriteRegisty.writeValue("myprogram", "program3", "sign2");
            ZHUCE = -1;
        }
    } catch (ParseException e) {
        e.printStackTrace();
    }
    if (ZHUCE > 0 || ReadWriteRegisty.readValue("myprogram", "program3").equals("sign1")) {
        try {
            ReadWriteRegisty.writeValue("myprogram", "program2", dateFormate.format(new Date()));
            SimpleDateFormat df2 = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            Date date_start = df2.parse(new Date().toLocaleString());
            Date date_end = df2.parse(ReadWriteRegisty.readValue("myprogram", "program1"));
            Calendar cal_start = Calendar.getInstance();
```







```

        cal_start.setTime(date_start);
        Calendar cal_end = Calendar.getInstance();
        cal_end.setTime(date_end);
        ZHUCE = days - getDifferenceDays(cal_start, cal_end);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
}
if (ZHUCE <= 0 || ReadWriteRegisty.readValue("myprograme", "programe3").
equals("sign2")) {
    ZHUCE = 0;
}
zhuce.setText(" (剩余" + ZHUCE + "天)");

```

在软件注册导航窗体中除了将试用的相关内容写入注册表之外,还要设计“继续”按钮的事件监听器,其关键代码如下:

```

b3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {

        if (b1.isSelected()) {
            setVisible(false);
            new RegisterFrame().setVisible(true);
        }
        if (b2.isSelected()) {
            if (ReadWriteRegisty.readValue("myprograme", "programe3").equals
("sign2")) {
                JOptionPane.showMessageDialog(mainFrame.this,
                    "软件已经试用到期,如果您想继续使用,请购买序列号进行注册使用");
                return;
            }
            if (ZHUCE == 0) {
                JOptionPane.showMessageDialog(mainFrame.this,
                    "软件已经试用到期,如果您想继续使用,请购买序列号进行注册使用");
                ReadWriteRegisty.writeValue("myprograme", "programe3", "sign2");
                return;
            }
        }
    }
});

```

## 9.4 软件注册窗体

### 9.4.1 功能概述

软件注册窗体为用户提供一个进行软件注册的功能,当用户在软件注册导航窗体中选择“我有一个注册码,我想使用该软件”的单选按钮,并单击“继续”按钮后,将进入软件注册窗体,该窗体的运行效果如图9.8所示。



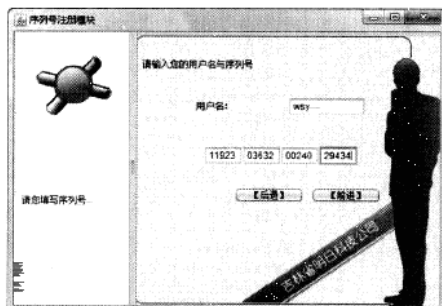


图 9.8 软件注册窗体

在软件注册窗体中，包括“用户名”文本框，还有 4 个用于填写注册码的文本框，同时该窗体还设置两个按钮分别为“后退”与“前进”按钮。“后退”按钮可以使该窗体返回到软件注册导航窗体中，而“前进”按钮将引导软件注册完成。

软件注册窗体与软件导航窗体的布局基本相同，都是由 JSplitPane 控件将该窗体进行左右分割设置的，在窗体的左侧放置介绍该窗体的内容面板，而窗体右侧放置主要完成注册功能的功能控件。其关键代码如下：

```
public RegisterFrame() {
    super();
    final JSplitPane splitPane = new JSplitPane();
    splitPane.setDividerLocation(150);
    JPanel rightPanel = new JPanel();
    ImageIcon icon = CreatecdIcon.add("11.jpg");
    jl = new JLabel();
    jl.setLayout(null);
    JLabel jl = new JLabel("<html><font=14> 请输入您的用户名与序列号</font></html>");
    jl.add(jl);
    jl.setBounds(10, 20, 160, 30);
    jl.setIcon(icon);
    JLabel userNameJ = new JLabel("用户名: ");
    final JTextField userNameT = new JTextField("wsy", 40);
    jl.add(userNameJ);
    jl.add(userNameT);
    userNameJ.setBounds(80, 80, 100, 25);
    userNameT.setBounds(200, 80, 100, 25);
    t1 = new JTextField(10);
    t2 = new JTextField(10);
    t3 = new JTextField(10);
    t4 = new JTextField(10);
    JTextJP = new JPanel();
    JTextJP.setBackground(new Color(234, 241, 247));
    JTextJP.setLayout(null);
    JTextJP.add(t1);
    JTextJP.add(t2);
    JTextJP.add(t3);
    JTextJP.add(t4);
    t1.setBounds(10, 15, 50, 25);
    t2.setBounds(60, 15, 50, 25);
    t3.setBounds(110, 15, 50, 25);
    t4.setBounds(160, 15, 50, 25);
    t1.addMouseListener(new JCopy());
    t2.addMouseListener(new JCopy());
}
```







```

t3.addMouseListener(new JCopy());
t4.addMouseListener(new JCopy());
j1.add(JTextJP);
JTextJP.setBounds(80, 130, 220, 50);
JButton back = new JButton("【后退】");
JButton forwards = new JButton("【前进】");
j1.add(back);
j1.add(forwards);
back.setBounds(130, 200, 90, 20);
forwards.setBounds(230, 200, 90, 20);
// 省略部分代码
j1.setBounds(0, 0, icon.getIconWidth(), icon.getIconHeight());
rightPanel.add(j1, new Integer(Integer.MIN_VALUE));
JPanel leftPanel = new JPanel();
leftPanel.setLayout(null);
leftPanel.setBackground(Color.WHITE);
JLabel image = new JLabel();
ImageIcon icon2 = ImageIcon.create("00002.jpg");
image.setIcon(icon2);
JLabel letter1 = new JLabel();
letter1.setText("<html>请您填写序列号</html>");
JLabel letter2 = new JLabel();
leftPanel.add(image);
image.setBounds(20, 20, 130, 100);
leftPanel.add(letter1);
letter1.setBounds(10, 120, 130, 200);
leftPanel.add(letter2);
letter2.setBounds(10, 200, 130, 50);
splitPane.setLeftComponent(leftPanel);
splitPane.setRightComponent(rightPanel);
getContentPane().add(splitPane);
setSize(new Dimension(570, 400));
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
m.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setTitle("序列号注册模块");
}

```

在软件注册窗体中, 主要包括“后退”与“前进”两个按钮, 为了让其完成一定功能, 需要增加监听器。“后退”按钮的事件监听器关键代码如下:

```

back.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        setVisible(false);
        m.setVisible(true);
    }
});

```

在上述代码中为了使软件注册导航窗体可见, 需要获取 mainFrame 类的对象, 这个对象在 RegisterFrame 类中作为全局变量在程序的最开始处已经被初始化。

“前进”按钮的事件监听中完成软件注册的功能, 这些功能主要包括验证注册码、限制注册用户的使用时间、根据注册机器的硬件信息保证软件使用的唯一性。在下面的章节中将详细讲解这些功能。







## 9.4.2 验证注册码

验证注册码的代码被封装在“前进”按钮的监听事件中，完成这一功能的步骤如下。

(1) 获取用户输入的验证码。

(2) 将用户名通过编码程序处理为密文，比较该密文与用户输入的验证码是否相同。  
在“前进”按钮事件监听器中验证注册码的关键代码如下：

```
forwards.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        s = userNameT.getText().trim();
        String t1Text = t1.getText().trim();
        String t2Text = t2.getText().trim();
        String t3Text = t3.getText().trim();
        String t4Text = t4.getText().trim();
        if (t1Text.length() != 0 && t2Text.length() != 0 && t3Text.length() != 0
            && t4Text.length() != 0) {
            if (s != null) {
                String[] a = RegisterMark.getRegister(s).split("-");
                if (!a[0].equals(t1Text) || !a[1].equals(t2Text)
                    || !a[2].equals(t3Text) || !a[3].equals(t4Text)) {
                    JOptionPane.showMessageDialog(RegisterFrame.this,
                        "用户名与注册码不匹配，请确定后重新输入");
                    t1.setText("");
                    t2.setText("");
                    t3.setText("");
                    t4.setText("");
                    return;
                } else {
                    String register = "";
                    for (int i = 0; i < a.length; i++) {
                        register += a[i];
                    }
                    BigInteger codeStringBigint = new encrypt().Execencrypt(
                        RegisterMark.getMAC(), register);
                    System.out.println(ReadWriteRegisty.readValue("myprograme",
                        "register1").equals("注册表中没有该值"));
                    if (ReadWriteRegisty.readValue("myprograme", "register1").
                        equals("注册表中没有该值")) {
                        ReadWriteRegisty.writeValue("myprograme", "register1",
                            String.valueOf(codeStringBigint));
                        ReadWriteRegisty.writeValue("myprograme", "registertime",
                            sf.format(new Date()));
                        ReadWriteRegisty.writeValue("myprograme",
                            "registertimetest", sf.format(new Date()));
                        ReadWriteRegisty.writeValue("myprograme", "programe3",
                            "sign3");
                    } else {
                        if (new decrypt().Execdecrypt().contains(RegisterMark.
                            getMAC())) {
                            JOptionPane.showMessageDialog(RegisterFrame.this,
                                "一个注册码只能在唯一一台计算机上使用");
                            return;
                        } else {
                            try {
                                if (new Date().compareTo(sf.parse(ReadWriteRegisty
                                    .readValue("myprograme", "registertimetest"))) < 0) {
                                    JOptionPane.showMessageDialog(RegisterFrame.this,
                                        "您的软件时间使用已经到期，如果希望继续使用，请注册");
                                    ReadWriteRegisty.writeValue("myprograme",
                                        "programe3", "sign1");
                                    return;
                                } else

```







```

        ReadWriteRegistry.writeValue("myprograme",
            "registertimetest",
            sf.format(new Date()));
    } catch (ParseException e1) {
        e1.printStackTrace();
    }
    try {
        Calendar cal_start = Calendar.getInstance();
        Calendar cal_end = Calendar.getInstance();
        cal_start.setTime(new Date());
        cal_end.setTime(sf.parse(ReadWriteRegistry.
            readValue("myprograme",
                "programe1")));
        System.out.println(cal_start.get(Calendar.YEAR));
        System.out.println(cal_end.get(Calendar.YEAR));
        if (new Date().before(sf.parse(ReadWriteRegistry.
            readValue("myprograme",
                "registertime")))) {
            JOptionPane.showMessageDialog(RegisterFrame.
                this, "您的软件时间使用已经到期, 如果希望继续使用, 请注册");
            ReadWriteRegistry.writeValue("myprograme",
                "programe3", "sign1");
            return;
        }
        if (cal_start.get(Calendar.YEAR) - cal_end.get(
            Calendar.YEAR) >= 1
            || cal_start.get(Calendar.YEAR) - cal_end.
                get(Calendar.YEAR) < 0) {
            ReadWriteRegistry.writeValue("myprograme",
                "programe3", "sign2");
            JOptionPane.showMessageDialog(RegisterFrame.this,
                "您的软件时间使用已经到期, 如果希望继续使用, 请注册");
            ReadWriteRegistry.writeValue("myprograme",
                "programe3", "sign1");
            return;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
JOptionPane.showMessageDialog(RegisterFrame.this, "软件注册成
功, 您可以使用该软件");
}
} else {
    JOptionPane.showMessageDialog(RegisterFrame.this, "请填写注册码与用户
    名");
}
}
});

```

在上述代码中使用到了 RegisterMark 类中的 getRegister() 方法, 用于获取用户名的密文, 其关键代码如下:

```

public static String getRegister(String s) {
    String s1 = "", s2 = ""; // 初始化两个字符串
    // 设定事件的格式化对象
    SimpleDateFormat sd = new SimpleDateFormat("yyyy");
    Date d = new Date(); // 初始化一个 Date 对象
    s += sd.format(d); // 将需要加密的字符串与当前年份连接
    for (int i = 0; i < s.length(); i++) { // 循环字符串的个数
        // 对字符串中每个字符进行处理, 并赋予 s1 字符串
        s1 += (int) s.charAt(i) * (i + 1);
    }
    // 根据 s1 字符串进行处理获取字符串 s2
    for (int i = 0; i < s1.length() / 5; i++) {
        if (i == s1.length() / 5 - 1)
            s2 += s1.substring(i * 5, (i + 1) * 5);
    }
}

```







```
        else
            s2 += s1.substring(i * 5, (i + 1) * 5) + "-";
    }
    return s2; // 将字符串 s2 返回
}
```

获取加密的字符串后，将该字符串以“-”字符进行分割，然后分别与软件注册窗体中的4个文本框进行比较，如果每个文本框的内容与该字符串的子串相同，那么就通过注册码的验证，将注册码返回给 register 变量，如果每个文本框的内容与该字符串的子串不同，那么注册码验证错误，将弹出错误提示对话框。

### 9.4.3 限制使用时间

用户注册后，不可能永久对该软件具有使用权，应该在序列号注册模块中为注册用户提供一个使用期限，所以要在注册代码中限制到期用户继续使用该软件。

首先需要判断用户注册使用时间是否到期，如果到期，更新注册表中 program3 键值的值为 sign1，判断用户注册试用时间是否到期可以使用当前时间与注册表中用户首次注册的时间进行比较，前者大于后者，说明注册到期，弹出提示对话框。

比较两个时间顺序可以使用 Date 类中的 compareTo() 方法，该方法的参数为 Date 类型，如果参数 Date 等于调用该方法的 Date，则返回值为 0，如果调用该方法的 Date 在参数 Date 之前，则返回小于 0 的值，如果调用该方法的 Date 在 Date 参数之后，则返回大于 0 的值。在调用该方法中可以可能会出现 NullPointerException 异常。

在“继续”按钮的事件监听器中判断注册用户是否到期的关键代码如下：

```
try {
    if (new Date().compareTo(sf.parse(ReadWriteRegistry.readValue("myprogram",
        "registertimetest")) < 0) {
        JOptionPane.showMessageDialog(RegisterFrame.this,
            "您的软件时间使用已经到期，如果希望继续使用，请注册");
        ReadWriteRegistry.writeValue("myprogram", "program3", "sign1");
        return;
    } else
        ReadWriteRegistry.writeValue("myprogram", "registertimetest", sf.
            format(new Date()));
} catch (ParseException e1) {
    e1.printStackTrace();
}
try {
    Calendar cal_start = Calendar.getInstance();
    Calendar cal_end = Calendar.getInstance();
    cal_start.setTime(new Date());
    cal_end.setTime(sf.parse(ReadWriteRegistry.readValue("myprogram",
        "program1")));
    System.out.println(cal_start.get(Calendar.YEAR));
    System.out.println(cal_end.get(Calendar.YEAR));
    if (new Date().before(sf.parse(ReadWriteRegistry.readValue("myprogram",
        "registertime")))) {
        JOptionPane.showMessageDialog(RegisterFrame.this,
            "您的软件时间使用已经到期，如果希望继续使用，请注册");
        ReadWriteRegistry.writeValue("myprogram", "program3", "sign1");
        return;
    }
    if (cal_start.get(Calendar.YEAR) - cal_end.get(Calendar.YEAR) >= 1
        || cal_start.get(Calendar.YEAR) - cal_end.get(Calendar.YEAR) < 0) {
        ReadWriteRegistry.writeValue("myprogram", "program3", "sign2");
        JOptionPane.showMessageDialog(RegisterFrame.this,
            "您的软件时间使用已经到期，如果希望继续使用，请注册");
        ReadWriteRegistry.writeValue("myprogram", "program3", "sign1");
    }
}
```







```

        return;
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

为了避免用户修改系统时间以获取更长的注册使用时间,与避免用户修改系统时间以获取更长的试用时间原理基本相同。在首次运行窗体时,将系统当前时间写入注册表中的 `registertimetest` 键值,然后在注册时间没有满的情况下更新当前时间作为 `registertimetest` 键值的值,这样用户每次注册的时候,将当前时间与注册表中 `registertimetest` 键值的时间比较,如果当前时间在该时间之前,说明用户修改了系统时间。

#### 9.4.4 保证使用唯一性

有时用户在两台硬件电脑上同时使用该软件,为了避免这种情况发生,在软件注册时,获取电脑的硬件信息、使用 RSA 算法进行加密,然后将密文写入配置文件中。每次用户进入注册窗体时都会读取配置文件的内容,首先对该内容进行解密操作,再获取当前机器的 MAC 地址与该配置文件的信息进行比较,如果相同说明用户是在同一台机器上注册的。如果用户在其他机器上运行该模块,配置文件中的硬件信息已被写入,程序会弹出“一个注册码只能在唯一一台机器上使用”的错误提示对话框。

实现根据注册机器的硬件信息保证软件使用唯一性功能的步骤如下。

(1) 首次注册时将该机器的 MAC 地址进行加密,将密文写入配置文件 `a.ini` 中。

(2) 在每次注册的过程中都获取 `a.ini` 文件中的密文进行解密,与当前 MAC 地址进行比较,判断软件是否在同一台机器上进行注册。

```

    BigInteger codeStringBigint = new encrypt().Execencrypt(RegisterMark.getMAC(),
register);
    System.out.println(ReadWriteRegisty.readValue("myprograme", "register1").
equals("注册表中没有该值"));
    if (ReadWriteRegisty.readValue("myprograme", "register1").equals("注册表中没有
该值")) {
        ReadWriteRegisty.writeValue("myprograme", "register1", String.valueOf
(codeStringBigint));
        ReadWriteRegisty.writeValue("myprograme", "registertime", sf.format(new
Date()));
        ReadWriteRegisty.writeValue("myprograme", "registertimetest", sf.format
(new Date()));
        ReadWriteRegisty.writeValue("myprograme", "programe3", "sign3");
    } else {
        if (new decrypt().Execdecrypt().contains(RegisterMark.getMAC())) {
            JOptionPane.showMessageDialog(RegisterFrame.this, "一个注册码只能在唯一
一台计算机上使用");
            return;
        }
    }
    // 省略部分代码
    在上述代码中使用了 encrypt 类中的 Execencrypt() 方法进行加密操作,其关键代码如下:
    public BigInteger Execencrypt(String str, String register) {
        BigInteger codeStringBigint = BigInteger.ZERO;
        try {
            File f = new File("a.ini");
            if (!f.exists()) {
                new genKey();
                // 读取公钥
                FileInputStream FIS = new FileInputStream("RSAPubKey.dat");
                ObjectInputStream OIS = new ObjectInputStream(FIS);
                RSAPublicKey RSAPK = (RSAPublicKey) OIS.readObject();
                // 得到两个公钥的参数

```





```
BigInteger e = RSAPK.getPublicExponent();
BigInteger n = RSAPK.getModulus();
// 将明文转为大整数
byte[] strByte = (str.concat(register)).getBytes("UTF8");
BigInteger m = new BigInteger(strByte);
// 进行加密操作
codeStringBigint = m.modPow(e, n);
// 保存密文
Properties pro = new Properties();
pro.put("register", String.valueOf(codeStringBigint));
try {
    // 将信息保存在 a.ini 文件中
    pro.store(new FileOutputStream("a.ini"), true);
    // 可以从 a.ini 中通过 Properties.get 方法读取配置信息
    pro.load(new FileInputStream("a.ini"));
} catch (Exception ex) {
    ex.printStackTrace();
}
} catch (Exception e) {
    e.printStackTrace();
}
return codeStringBigint;
}
```

## 9.5 注册机实现

### 9.5.1 功能概述

注册机是用于生成注册码的窗体，其运行效果如图 9.9 所示。

在注册机窗体中，输入 3 位用户名，然后单击“生成注册码”按钮，在注 4 个文本框中将分别放置 4 组 5 位的注册码，它将用于在软件注册时使用。同时还为 4 个注册码文本框添加了鼠标单击事件，当用户在 4 组文本框中任意文本框中单击鼠标右键，窗体都会在鼠标单击位置弹出菜单。运行效果如图 9.10 所示。

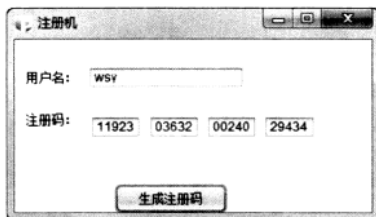


图 9.9 注册机窗体

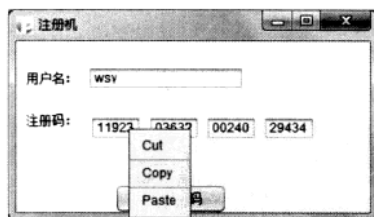


图 9.10 注册机窗体



268



注册机窗体的界面设计比较简单，首先为整个窗体添加布局管理器，这里使用绝对布局，然后将标签、文本框及按钮放入窗体中合适的位置，其关键代码如下：

```
public RegisterTradeMark() {
    super();
    getContentPane().setBackground(new Color(240, 255, 255));
    try {
        UIManager.setLookAndFeel(new com.sun.java.swing.plaf.nimbus.
            NimbusLookAndFeel());
    } catch (UnsupportedLookAndFeelException e) {
        e.printStackTrace();
    }
}
```





```

    }
    setTitle("注册机");
    getContentPane().setLayout(null);
    final JLabel label = new JLabel();
    label.setBounds(10, 26, 52, 18);
    label.setText("用户名: ");
    getContentPane().add(label);
    userName = new JTextField();
    userName.setBounds(68, 24, 147, 22);
    getContentPane().add(userName);
    final JLabel label_1 = new JLabel();
    label_1.setBounds(10, 66, 52, 18);
    label_1.setText("注册码: ");
    getContentPane().add(label_1);
    final JPanel panel = new JPanel();
    panel.setBackground(new Color(240, 255, 255));
    panel.setLayout(null);
    panel.setBounds(60, 50, 261, 68);
    getContentPane().add(panel);
    t1 = new JTextField();
    t1.setBounds(10, 21, 48, 22);
    panel.add(t1);
    t2 = new JTextField();
    t2.setBounds(65, 21, 48, 22);
    panel.add(t2);
    t3 = new JTextField();
    t3.setBounds(119, 21, 48, 22);
    panel.add(t3);
    t4 = new JTextField();
    t4.setBounds(173, 21, 49, 22);
    panel.add(t4);
    final JPopupMenu popup = new JPopupMenu();
    JMenuItem itemcut = new JMenuItem("Cut");
    popup.add(itemcut);
    popup.addSeparator();
    JMenuItem itemcopy = new JMenuItem("Copy");
    popup.add(itemcopy);
    popup.addSeparator();
    JMenuItem itempaste = new JMenuItem("Paste");
    popup.add(itempaste);
    t1.addMouseListener(new PopAction(popup));
    t2.addMouseListener(new PopAction(popup));
    t3.addMouseListener(new PopAction(popup));
    t4.addMouseListener(new PopAction(popup));
    // 设置整个页面组件焦点从后向前
    this.setFocusTraversalKeys(KeyboardFocusManager.BACKWARD_TRAVERSAL_KEYS,
null);
    // 省略部分代码
    button.setText("生成注册码");
    button.setBounds(93, 135, 106, 28);
    getContentPane().add(button);
    setSize(new Dimension(350, 200));
    setIconImage(CreatecdIcon.add("ico.jpg").getImage());
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}

```

### 9.5.2 生成注册码

生成注册码的功能应该在“生成注册码”按钮的事件监听器中完成，在按钮监听事件中首先获取用户名，然后根据用户名字符串经过一定的处理转为4组5位的注册码，其关键代码如下：

```

button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(final ActionEvent arg0) {

```





```
String userNameT = userName.getText().trim();
String s = RegisterMark.getRegister(userNameT);
String a[] = s.split("-");
t1.setText(a[0]);
t2.setText(a[1]);
t3.setText(a[2]);
t4.setText(a[3]);
}
});
```

在上述代码中使用 RegisterMark 类中的 getRegister()方法对用户名进行加密，其关键代码如下：

```
public static String getRegister(String s) {
    String s1 = "", s2 = "";
    SimpleDateFormat sd = new SimpleDateFormat("yyyy");
    Date d = new Date();
    s += sd.format(d);
    for (int i = 0; i < s.length(); i++) {
        s1 += s.charAt(i) * (i + 1);
    }
    for (int i = 0; i < s1.length() / 5; i++) {
        if (i == s1.length() / 5 - 1)
            s2 += s1.substring(i * 5, (i + 1) * 5);
        else
            s2 += s1.substring(i * 5, (i + 1) * 5) + "-";
    }
    return s2;
}
```





# 第 10 章

---

## PDF 查看模块

(Swing+PDF Render 实现)

谈到 PDF 查看程序时，大多数人都会想到当前比较流行的几款 PDF 查看程序，比如 Adobe Reader 电子阅读器、超星电子阅读器等，这两个款软件支持 PDF、PNG 等类型文档的阅读。不仅支持各种类型文档的阅读，同时也为用户在阅读该类文档时提供了较好的服务，比如书签、缩位图导航、页码定位、文档放大缩小、文档快照、以指定格式导出 PDF 文档等功能，从现在的情况来看，开源的 PDF 电子阅读器软件以国外居多，并且对中文支持并不好，本章将开发一个功能完备、强大的 PDF 查看模块，并且向读者展示整个 PDF 查看模块开发的全部过程。通过本章的学习，读者能够学到：

- » 指定翻页
- » 缩放页面、全屏显示
- » 打开 PDF 文档
- » 缩位图导航
- » 书签导航



## 10.1 PDF 查看模块概述

### 10.1.1 模块概述

本模块主要实现读取 PDF 文档的功能,通过使用本模块用户可以选择需要读取的 PDF 文档进行阅读,在阅读的过程中,为了方便用户的使用,提供了书签、文档缩位图的导航功能,用户可以根据书签或缩位图的方式来指定需要阅读某页文档。本模块还为用户提供了打印、打印页面设置等功能,除此之外本模块还具有对文档逐一翻页、指定具体页面、放大或缩小指定页面、全屏的功能,笔者还为本模块添加了文档自动滚动与抓手等功能,这些功能可以确保用户更方便、快捷地读取 PDF 文档。

### 10.1.2 功能结构

在本模块中,主要包括 4 大部分,分别为 PDF 文档管理、文档导航、系统管理、PDF 文档读取管理,其中 PDF 文档管理部分主要包括打开 PDF 文档、打印 PDF 文档、页面设置等功能,而文档导航部分主要包括缩位图导航、书签导航及指定翻页等功能,系统管理主要包括退出系统的功能实现,PDF 文档读取管理部分主要包括文档缩放、读取 PDF 文档、文档抓手功能、文档自动滚动及全屏显示功能。PDF 查看模块的功能结构图如图 10.1 所示。

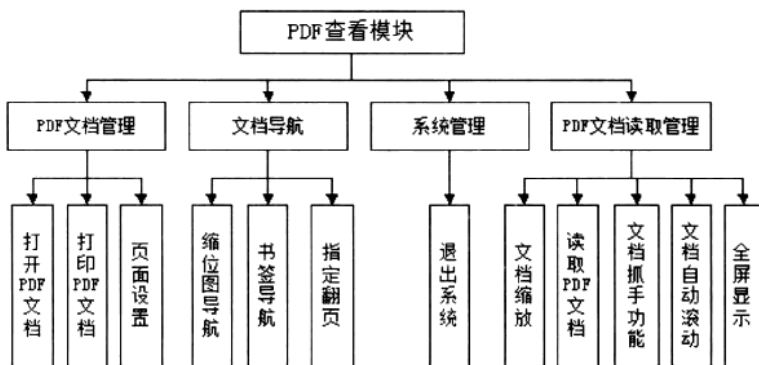


图 10.1 PDF 查看模块功能结构图



### 10.1.3 程序预览

下面列举出本模块中的几个典型窗体,其他窗体可以参见光盘中的源程序,本模块的主窗体运行效果如图 10.2 所示,它主要用于承载 PDF 查看程序的整个功能。



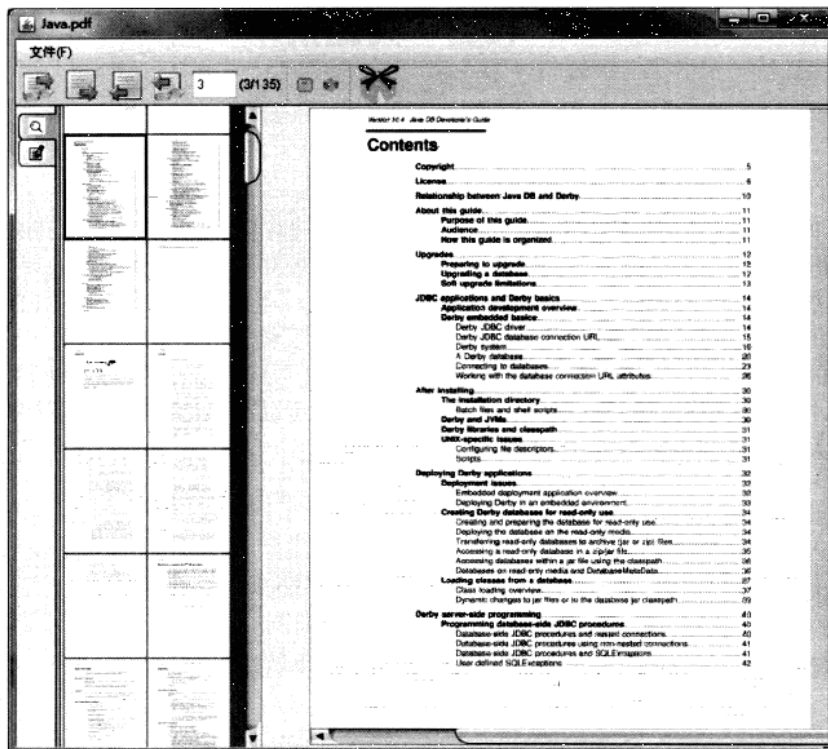


图 10.2 PDF 查看模块主窗体

缩位图导航如图 10.3 所示, 书签导航如图 10.4 所示, 这两个功能主要为整个文档起到一个导航的作用和指定阅读文档中的当前页面功能。

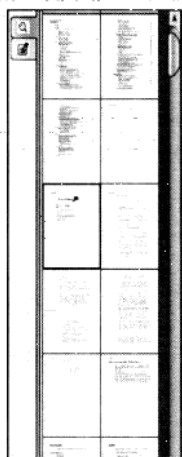


图 10.3 缩位图导航

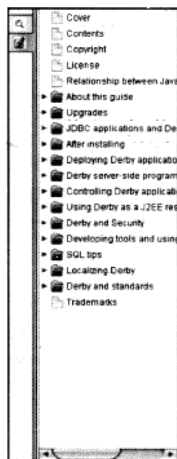


图 10.4 书签导航





## 10.2 关键技术

### 10.2.1 PDF Render 组件技术

PDF Render 组件是一款与 PDF 文档进行交互的开源组件，它只可以在 JDK1.5 以上的版本下运行，它较其他读取 PDF 文档的组件功能比较并不是很强大，但在程序中使用起来却很方便，其中与 PDF 文档进行交互的 Java 开源组件主要包括 PDFBox、XPDF、iText、PJX 及 PDF Render 组件等，iText 组件多用于实现 PDF 文档写入功能，而 XPDF 软件读取 PDF 文档的方法只是运行 XPDF 这个软件，它并不是基于 Java 语言的组件，虽然 PDFBox 功能强大，但是 PDFBox 对中文的支持也并不是很好，所以在这里笔者选择了相对功能并不是很强大的 PDF Render 组件来开发 PDF 查看模块。

PDF Render 组件主要具有以下几个特点：

- ☐ 可以与 PDF 文档进行交互
- ☐ 绘制 PDF 文档导航
- ☐ 以 3D 模式显示 PDF 文档
- ☐ 将 PDF 文档转换为图片格式
- ☐ 笔者正是基于以上这些特点进行 PDF 查看模块的开发。

### 10.2.2 实现 PDF 文档缩放

PDF 文档的缩放在 PDF 阅读器中是比较常见的功能，用户可以根据自己的需求调整读取页面的大小，在实现这个功能的过程中笔者使用 PDF Render 组件将需要读取的 PDF 文档转变为图片格式，然后相应放大或缩小该图片并将其显示在面板上。下面详细介绍缩放 PDF 文档是如何实现的。

#### 1. PDF 文档放大

在上文中提到，若想实现 PDF 文档放大需要将 PDF 文档转变为图片格式，这是 PDF Render 组件的一个功能，可以使用 `com.sun.pdfview.PDFPage.java` 类中的 `getImage()` 方法将某个 PDF 页面转变为指定大小的图片。



PDFPage.java 类中包括两个 `getImage()` 方法，这两个方法是重载方法，参数



个数不同。在这里使用 `public Image getImage(int width, int height, Rectangle2D clip, ImageObserver observer, boolean drawbg, boolean wait)` 方法。

当将某页文档转变为指定大小的图片后就可以将该图片放置在标签中，然后将该标签放置在面板中进行显示。在模块中将 PDF 文档放大 2 倍，运行结果如图 10.5 和图 10.6 所示。



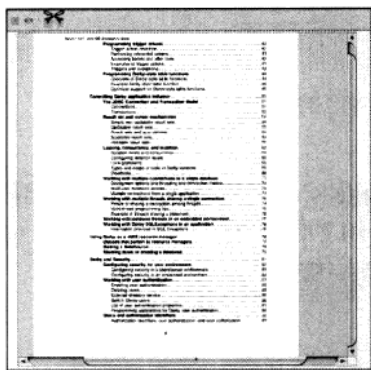


图 10.5 原大小 PDF 文档

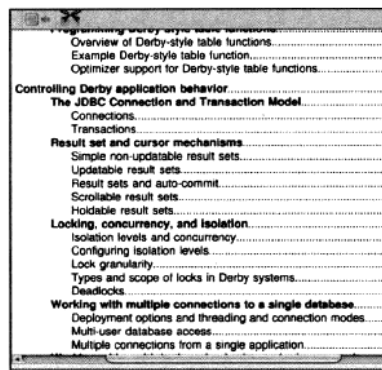


图 10.6 放大 PDF 文档

在程序中实现这个功能的步骤如下。

(1) 在工具栏中初始化一个放大按钮，在初始化的同时指定这个按钮将触发的 Action。该步骤的实现代码如下：

```
bigButton = new JButton(BigAction);           // 初始化放大按钮
toolBar.add(bigButton);                       // 将该按钮添加到工具栏上
bigButton.setEnabled(false);                  // 使该按钮不可用
bigButton.setToolTipText("页面放大");         // 为该按钮设置提示文字
bigButton.setIcon(CreatecdIcon.add("big.gif")); // 为该按钮设置图标
```

(2) 编写放大按钮的 Action 对象 BigAction。

BigAction 继承了 AbstractAction 类，由于这个类是一个抽象类，所以在继承该类的同时需要重写 actionPerformed() 方法，在这里使用了匿名内部类的形式。关键代码如下：

```
Action BigAction=new AbstractAction(){
    public void actionPerformed(ActionEvent arg0) {
        // 实例化一个 PDFPage 对象
        PDFPage page = pdfFile.getPage(curpage);
        Rectangle rect = new Rectangle(0,0, (int)page.getBBox().getWidth(),
                                         (int)page.getBBox().getHeight());
        // 生成一个图片
        Image img = page.getImage(           // 该方法返回一个包含 PDF 文档数据的图片
            rect.width*2, rect.height*2, // 设置该图片的大小，这里指定是原图片大小的两倍
            rect,                          // 指定一个矩形放置该图片
            null,
            // 这个布尔型变量指定图片的背景，如果为 true，说明背景为白色
            true,
            // 该参数指定须等待该图片绘制完成后运行其他线程
            true
        );
        if(jpmain!=null){
            jpmain.removeAll();           // 删除以往指定面板上的 PDF 数据
        }
        contentPanel.setViewportView(jpmain);
        contentPanel.setBounds(230, 60, 752, 600);
        // 使该面板中的鼠标具有抓手功能
        jpmain.addMouseListener(new JPMouseAction());
        jpmain.addMouseMotionListener(new JPMouseMotionAction());
        // 将该图片放置在 JLabel 上然后添加到面板中
        jpmain.add(new JLabel(new ImageIcon(img)));
        validate();
    }
}
```





```

    }
    repaint();
}

```

通过上述两个步骤就可以实现 PDF 文档放大两倍的功能。

## 2. PDF 文档缩小

PDF 文档缩小与 PDF 文档放大功能实现基本相同，只是将相应的 PDF 文档图片缩小 1/2，PDF 文档缩小的功能在程序中实现结果如图 10.7 所示。

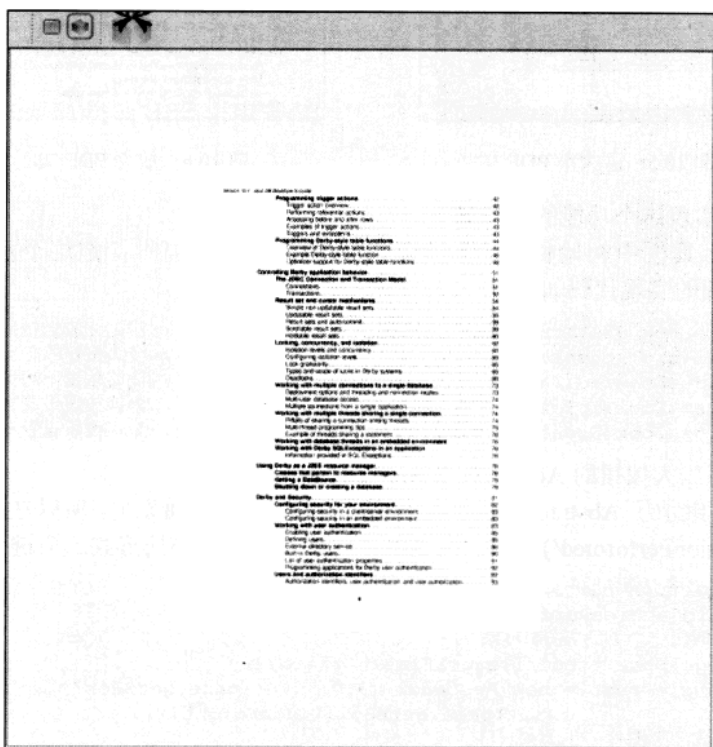


图 10.7 缩小 PDF 文档

PDF 文档缩小功能的实现步骤如下。

(1) 在工具栏中初始化一个缩小按钮，为该按钮添加图标，指定提示文字及为该按钮添加触发事件的 Action，关键代码如下：

```

smallButton = new JButton(SmallAction); // 添加缩小按钮并为该按钮添加指定的 Action
smallButton.setEnabled(false);          // 设置该按钮不可用
smallButton.setToolTipText("页面缩小"); // 为该按钮设置提示文字
smallButton.setIcon(CreatecdIcon.add("small.gif")); // 为该按钮设置图标
toolBar.add(smallButton);               // 将该按钮添加到工具栏中

```

(2) 编写 SmallAction，SmallAction 同样也继承了 AbstractAction 类，该类也需要重写 actionPerformed() 方法，关键代码如下：

```

Action SmallAction=new AbstractAction(){
    public void actionPerformed(ActionEvent arg0) {
        PDFPage page = pdfFile.getPage(curpage); // 实例化一个 PDFPage 对象
        Rectangle rect = new Rectangle(0,0,

```







```

        (int)page.getBBox().getWidth(),
        (int)page.getBBox().getHeight());
    // 构建一个图片
    Image img = page.getImage(
        // 设置该图片的大小, 这里指定图片大小是原大小的 1/2
        rect.width/2, rect.height/2,
        rect,
        null,
        true,
        true
    );
    if(jpmain!=null){
        jpmain.removeAll();
    }
    contentPanel.setViewportView(jpmain);
    contentPanel.setBounds(230, 60, 600, 600);
    // 将定义完成的图片放置在标签上, 然后放置在相应面板中
    jpmain.add(new JLabel(new ImageIcon(img)));
    validate();
    repaint();
}
};

```

### 10.2.3 实现 PDF 文档分页

PDF 文档的分页是比较重要的功能,它使用户在阅读的过程中可以查看指定页面而不必从头翻起,其中笔者主要为 PDF 文档设计了 3 种分页功能,包括用户页面定位、上一页、下一页、翻页及首页与尾页翻页功能。而这 3 种功能的实现都是围绕着 PDF Render 组件的 `pageChangeListener` 接口实现的,该接口提供了一个 `gotoPage()` 方法,可以在程序中重写该方法实现 PDF 文档分页功能。

#### 1. 用户页码定位

用户页码定位功能允许用户在指定文本框中填写数字,然后按下键盘【Enter】键就可以翻到指定页面进行阅读,同时主窗体导航索引也会指定在相应的页面,它在模块中实现如图 10.8 所示。

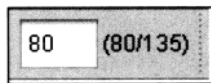


图 10.8 用户页码定位

实现该功能的步骤如下。

(1) 创建文本框,并为该文本框添加事件监听器,关键代码如下:

```

pageField = new JTextField("-", 3); // 初始化一个 JTextField 组件
pageField.setPreferredSize(new Dimension(10,30)); // 设置该文本框的大小
pageField.setEditable(false); // 设置该组件不可用
pageField.addActionListener(new ActionListener() { // 为该文本框添加事件
    public void actionPerformed(ActionEvent evt) { // 重写 actionPerformed() 方法
        int pagenum = -1; // 指定初始页码
        try {
            // 获取用户填写页码
            pagenum = Integer.parseInt(pageField.getText()) - 1;
        } catch (NumberFormatException e) {}
    }
});

```



277





```
    }
    catch (NumberFormatException nfe) {
    }
    if (pagenum >= pdffile.getNumPages()) { // 如果用户指定的页码大于文档的总页码
        pagenum = pdffile.getNumPages() - 1; // 将文档总页码指定赋予 pagenum 变量
    }
    if (pagenum >= 0) { // 如果 pagenum 变量的值不为-1
        if (pagenum != curpage) { // 如果当前页码不等于 pagenum 变量的值
            gotoPage(pagenum); // 调用 gotoPage() 方法进入指定的页面
        }
    } else {
        // 如果 pagenum 变量的值等于-1, 将当前的页码写入文本框中
        pageField.setText(String.valueOf(curpage));
    }
}
});
```



在上述代码中, pdffile 变量是 PDFFile 类对象, 它是 PDF Render 组件中的类, 它的主要作用就是提供 PDF 文档的相关信息。

(2) 使主窗体 MainFrame.java 类实现 PageChangeListener 接口, 在 MainFrame 类中重写 gotoPage() 方法, 关键代码如下:

```
public void gotoPage(int pagenum) {
    if (pagenum < 0) { // 如果参数 pagenum 小于 0
        pagenum = 0; // 将 0 赋予 pagenum 变量
    } // 如果参数 pagenum 大于等于 PDF 文档的最大页码
    else if (pagenum >= pdffile.getNumPages()) {
        pagenum = pdffile.getNumPages() - 1; // 将 PDF 文档的最大页码赋予 pagenum 变量中
    }
    forceGotoPage(pagenum); // 自定义 forceGotoPage() 方法, 进入指定页面操作
}
forceGotoPage() 方法的关键代码如下:
public void forceGotoPage(int pagenum) {
    if (pagenum <= 0) {
        pagenum = 0;
    } else if (pagenum >= pdffile.getNumPages()) {
        pagenum = pdffile.getNumPages() - 1;
    }
    curpage = pagenum;
    // 更新文本框的值
    pageField.setText(String.valueOf(curpage + 1));
    // 更新标签的指定页码
    jl.setText("(" + (curpage + 1) + "/" + pdffile.getNumPages() + ")");
    // 指定阅读器内容
    PDFPage pg = pdffile.getPage(pagenum + 1);
    if (jpmain != null) {
        jpmain.removeAll(); // 将 jpmain 面板上的内容删除
    }
    jpmain.add(jp); // 将承载着 PDF 文档中的面板添加到 jpmain 面板中
    validate();
    if (fspp != null) { // 如果全屏面板不为空
        fspp.showPage(pg); // 将 PDF 文档放置在全屏面板上
        fspp.requestFocus(); // 并使全屏面板获取焦点
    } else {
        // 如果全屏面板为空将 PDF 文档放置在 jp 面板中, jp 面板为全局变量, 它是 PagePanel 对象
        jp.showPage(pg);
    }
}
```







```

        jp.requestFocus(); // 并使 jp 面板获取焦点
    }
    thumbs.pageShown(pagenum); // 设置缩位图导航面板指向的页码
}

```

通过以上两个步骤, 就实现了本模块页码定位的功能。

## 2. 翻页功能

翻页是另一种分页形式的体现, 它可以为用户提供上一页、下一页功能按钮, 该功能在模块中的使用如图 10.9 所示。

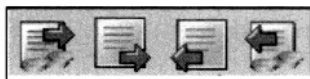


图 10.9 翻页功能

翻页功能的实现步骤如下。

(1) 在工具栏中放置“上一页”、“下一页”翻页按钮, 关键代码如下:

```

nextButton = new JButton(nextAction); // 初始化“下一页”按钮
nextButton.setEnabled(false); // 设置该按钮不可用
nextButton.setIcon(CreatecdIcon.add("next.gif")); // 为该按钮设置图标
nextButton.setToolTipText("下一页"); // 为该按钮设置提示文字
nextButton.setHideActionText(true); // 使该按钮不显示文本
toolBar.add(nextButton); // 将该按钮添加到工具栏中
backButton = new JButton(prevAction); // 初始化“上一页”按钮
backButton.setEnabled(false); // 设置该按钮不可用
backButton.setIcon(CreatecdIcon.add("back.gif")); // 为该按钮设置图标
backButton.setToolTipText("上一页"); // 为该按钮设置提示文字
backButton.setHideActionText(true); // 使该按钮不显示文本
toolBar.add(backButton); // 将该按钮添加到工具栏中

```

(2) 编写“上一页”、“下一页”按钮的 Action, 关键代码如下:

```

Action nextAction = new AbstractAction() {
    public void actionPerformed(ActionEvent evt) { // 重写 actionPerformed() 方法
        doNext(); // 调用 doNext() 方法, 用于实现下一页翻页功能
    }
};
Action prevAction = new AbstractAction() {
    public void actionPerformed(ActionEvent evt) { // 重写 actionPerformed() 方法
        doPrev(); // 调用 doNext() 方法, 用于实现上一页翻页功能
    }
};

```

其中 doNext() 与 doPrev() 方法用于完成文档的上一页与下一页的翻页功能, 这两个方法的关键代码如下:

```

public void doNext() {
    gotoPage(curpage + 1); // 调用 gotoPage() 方法使 PDF 文档翻到下一页
}
public void doPrev() {
    gotoPage(curpage - 1); // 调用 gotoPage() 方法使 PDF 文档翻到上一页
}

```

在上述代码中可以看到, 文档具体指定的页数都是基于 gotoPage() 方法的参数而决定的, 其中 curpage 是一个 int 型的全局变量, 它用于表示当前的页码。





### 3. 首页、尾页定位

首页与尾页定位与上一页、下一页翻页功能实现原理基本相同，都是重写 gotoPage() 方法指定链接到第一页或最后一页。

在首页、尾页定位按钮的 Action 中定义了 doFirst() 与 doLast() 方法，在这两个方法中调用 gotoPage() 方法，关键代码如下：

```
public void doFirst() {  
    gotoPage(0); // 链接到 PDF 文档第一页  
}  
public void doLast() {  
    gotoPage(pdfFile.getNumPages() - 1); // 链接到 PDF 文档最后一页  
}
```

通过调用重写的 gotoPage() 方法就可以实现首页、尾页定位的功能。



pdfFile 为 PDFFile 类对象，它调用 getNumPages() 方法可以返回 PDF 文档的总页码。

## 10.2.4 实现 PDF 文档打印、页面设置

### 1. 实现 PDF 文档打印功能

一个软件不可缺少的部分就是打印功能，在本模块中也设置这两个功能。当用户选择菜单栏中的“文件”\“打印”菜单项后，会弹出一个提示添加打印机的，如图 10.10 所示。

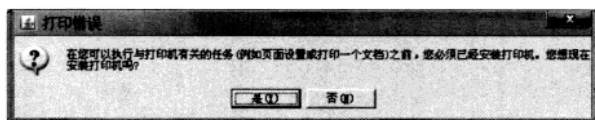


图 10.10 提示添加打印机

当用户在如图 10.10 所示的对话框中单击“是”按钮时，会弹出添加打印机向导对话框，用户可以自行添加打印机。当用户添加打印机成功后，就可以对相应页面进行打印了。实现打印功能的步骤如下。

(1) 在菜单栏中添加“打印”菜单项，然后为该菜单项添加事件监听器，关键代码如下：

```
final JMenuItem newItemMenuItem_3 = new JMenuItem(); //实例化一个 JMenuItem 对象  
newItemMenuItem_3.setText("打印"); //为该菜单项设置名称  
//为该菜单项添加单击监听事件  
newItemMenuItem_3.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) { //重写 actionPerformed() 方法  
        doPrint(); //调用 doPrint() 方法  
    }  
});  
newItemMenuItem1.add(newItemMenuItem_3); //将该菜单项添加到菜单中
```

在 doPrint() 方法中，主要完成对 PDF 文档的打印功能，它的关键代码如下：

```
public void doPrint() {
```







```
// 启动一个打印作业, 调用 getPrinterJob() 方法获取一个打印作业对象
PrinterJob pjob = PrinterJob.getPrinterJob();
if (docName != null && docName.length() != 0) { // 如果用户在弹出文件对话框中选择了文件
    pjob.setJobName(docName); // 设置打印的文档名称
    Book book = new Book(); // 初始化 Book 对象
    // 初始化 PDFPrintPage 对象
    PDFPrintPage pages = new PDFPrintPage(pdfFile);
    // 将整个 PDF 文档页面追加到 Book 的尾部
    book.append(pages, pformat, pdfFile.getNumPages());
    pjob.setPageable(book);
    // printDialog() 方法向用户提供一个打印对话框, 如果用户选择“是”, 则该方法返回 true,
    // 否则返回 false
    if (pjob.printDialog()) {
        // 启动打印线程
        new PrintThread(pages, pjob).start();
    }
} else
    // 如果没有在文件对话框中选择文件, 则弹出错误提示对话框
    JOptionPane.showMessageDialog(MainFrame.this, "请选择 PDF 文档后打印");
}
```

(2) 编写实现打印线程的类, 这里以内部类的形式进行表示, 它的关键代码如下:

```
class PrintThread extends Thread { // 自定义打印线程类, 它继承 Thread 类
    PDFPrintPage ptPages; // 定义一个 PDFPrintPage 对象
    PrinterJob ptPjob; // 定义一个打印作业
    // 该类的构造方法, 参数为打印的文档与打印作业
    public PrintThread(PDFPrintPage pages, PrinterJob pjob) {
        ptPages = pages;
        ptPjob = pjob;
    }
    public void run() { // 重写 run() 方法
        try {
            ptPages.show(ptPjob); // 弹出打印对话框
            ptPjob.print(); // 进行打印操作
        } catch (PrinterException pe) { // 在捕捉异常块中弹出错误提示对话框
            JOptionPane.showMessageDialog(MainFrame.this,
                "Printing Error: " + pe.getMessage(),
                "Print Aborted",
                JOptionPane.ERROR_MESSAGE);
        }
        ptPages.hide(); // 关闭打印进程
    }
}
```



在执行打印操作时, PrinterJob 类中的 print() 方法会连续不断地被调用, 因为打印作业并不知道用户需要打印文档的页码, 而是不断地调用 print() 方法, 只要该方法的返回值为 Printable.PAGE\_EXISTS, 那么打印作业就会不断地产生出页, 而如果 print() 方法的返回值为 Printable.NO\_SUCH\_PAGE 时, 打印作业就停止。

## 2. 实现 PDF 文档页面设置功能

该功能与打印功能相似, 当用户选择菜单栏中的“文件”、“页面设置”, 如果已经成功连接打印机将弹出一个“页面设置”对话框, 如图 10.11 所示。

实现该功能的步骤如下。



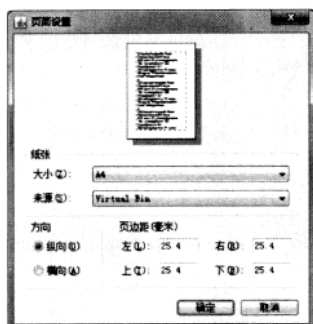


图 10.11 “页面设置”对话框

(1) 在菜单栏上添加“页面设置”菜单项，然后为该菜单项设置触发事件。由于该部分实现过程与“打印”菜单项的实现原理一致，所以这里不再赘述。读者可以在光盘查询相应源代码。

(2) 在该菜单项触发事件中调用实现 PDF 文档页面设置的方法 `doPageSetup()` 方法，在该方法中显示一个允许修改 `PageFormat` 实例的对话框，其中 `PageFormat` 类是一个用于描述打印页面大小和方向的类，该方法的关键代码如下：

```
public void doPageSetup() {
    PrinterJob pjob = PrinterJob.getPrinterJob(); // 实例化一个打印作业
    // 显示一个允许修改 PageFormat 实例的对话框
    pformat = pjob.pageDialog(pformat);
}
```

## 10.2.5 实现 PDF 文档自动滚动功能

PDF 文档自动滚屏功能主要是获取承载 PDF 文档内容的滚动面板对象，然后使用该对象调用 `getVerticalScrollBar()` 方法获取这个滚动面板的纵向滚动条对象，再使用该对象调用 `setValue()` 方法为该滚动条赋值。

为了实现自动滚动的功能，需要使用 `java.swing.Timer` 类制作一个定时器，然后将改变滚动条值的动作监听器作为定时器的参数。

实现 PDF 文档自动滚动功能的步骤如下：

(1) 制作一个定时器，它的参数为 `ActionListener` 对象。这个定时器起到每 0.5 秒触发一次动作事件的作用，它的关键代码如下：

```
activityMonitor=new Timer(500,new ActionListener(){ // 实例化一个 Timer 对象
// 在参数中存在一个 ActionListener 实例，重写 actionPerformed()方法
public void actionPerformed(ActionEvent arg0) {
    int current=activity.getCurrent(); // 获取一个值作为纵向滚动条的当前位置
    contentPanel.getVerticalScrollBar().setValue(current);
    if (current==activity.getTarget()){ // 当当前值到达预设值时
        activityMonitor.stop(); // 将当前定时器关闭
    }
}
});
```

上述代码中的 `activity` 对象是 `SimulatedActivity` 类的一个实例，该类实现 `current` 值每秒增加 10 倍的线程。它设置一个目标值，这个值是滚动条的最底端的值，可以使用对滚动面板调用 `getVerticalScrollBar().getMaximum()` 方法来获取，`SimulatedActivity` 类的关键代码如下：

```
// SimulatedActivity 类实现了 Runnable 接口
class SimulatedActivity implements Runnable {
    private volatile int current; // 滚动条当前值
    private int target; // 滚动条目标值
```





(2) 当定时器与设定滚动条值的线程都定义完成后, 将在打开 PDF 文档的方法中开启定时器与设置滚动条值的线程, 具体代码如下:

```
for(int i=1;i<10;i++){ // 这里只显示 PDF 文档的前 10 页
    contentPanel.setViewportView(jpmain); // 将一个面板放置滚动面板中
    // 实例化一个 SimulateActivity 类对象, 并将参数设置为滚动条最底端的值
    activity=new SimulateActivity(contentPanel.getVerticalScrollBar().
        getMaximum());
    new Thread(activity).start(); // 开启设置滚动条值的线程
    activityMonitor.start(); // 开启定时器
    .....// 省略部分代码
}
```

PDF 文档抓手功能在用户阅读的过程中使用起来十分方便,这个功能在程序中的使用如图 10.12 所示。



图 10.12 抓手功能



当用户单击承载 PDF 文档的面板时，鼠标由箭头形状变为手形，拖曳鼠标也可以同时移动文档页面，这就是抓手功能的具体体现。

抓手功能在程序中的具体实现步骤如下。

(1) 编写使鼠标由箭头变为手形的事件监听器。这个事件监听是鼠标点击事件接口 `MouseListener`，但是考虑到实现该接口需要实现 6 个方法，所以这个事件监听器继承了与该接口对应的适配器类 `MouseAdapter` 类，并在其中重写 `mousePressed()` 与 `mouseReleased()` 方法来描述鼠标按下与释放时进行的操作，实现这个监听器的关键代码如下：

```
// JPMouseAction 监听器继承了 MouseAdapter 类
private final class JPMouseAction extends java.awt.event.MouseAdapter {
    // 重写 mouseReleased() 方法
    public void mouseReleased(java.awt.event.MouseEvent e) {
        isDragged = false;
        setCursor(new Cursor(Cursor.DEFAULT_CURSOR)); // 将鼠标设置为默认状态
    }
    // 重写 mousePressed() 方法
    public void mousePressed(java.awt.event.MouseEvent e) {
        tmp = new Point(e.getX(), e.getY()); // 记下鼠标刚开始按下时的位置
        isDragged = true;
        setCursor(new Cursor(Cursor.HAND_CURSOR)); // 将鼠标设置为手形
    }
}
```

(2) 编写可以使鼠标拖曳窗体页面的事件监听器，该监听器继承了 `MouseMotionAdapter` 类，这个类为用户拖动鼠标提供了依据，在监听器中重写 `mouseDragged()` 方法，提供鼠标拖曳时进行的一系列操作，该监听器的关键代码如下：

```
private final class JPMouseMotionAction extends java.awt.event.
    MouseMotionAdapter {
    public void mouseDragged(java.awt.event.MouseEvent e) {
        if(isDragged) { // 当鼠标已经由箭头变为手形时
            // 在拖曳事件中不断记下改变位置
            loc = new Point(jp.getLocation().x + e.getX() - tmp.x,
                jp.getLocation().y + e.getY() - tmp.y);
            // 将承载着 PDF 文档的面板定位在鼠标拖曳的位置上
            jp.setLocation(loc);
        }
    }
}
```

(3) 可以在使放置 PDF 文档的主面板中调用上述两个事件监听器。

```
for(int i=1;i<10;i++){
    .....// 省略部分代码
    PDFPage page = pdfFile.getPage(i); // 打开每一页 PDF 文档
    PagePanel jp2 = new PagePanel(); // 实例化一个放置文档的面板
    jp2.addMouseListener(new JPMouseAction()); // 为该面板添加鼠标单击事件监听器
    // 为该面板添加鼠标拖曳事件监听器
    jp2.addMouseMotionListener(new JPMouseMotionAction());
    jpmain.add(jp2); // 将该面板添加到主面板上
    validate(); // 验证该容器的所有组件
    jp2.showPage(page); // 在面板上放置 PDF 文档
}
```







## 10.3 主窗体

### 10.3.1 功能概述

主窗体主要包括菜单栏、工具栏、左侧索引面板及右侧界面的实现,这种主窗体设计模式是比较普遍的一种设计模式,其中,菜单用于打开文档、打印文档、对文档进行页面设置及关闭阅读器;工具栏用于浏览页面、显示指定页面、最大化页面、最小化页面及全屏显示页面;左侧索引面板主要用于显示缩位图导航及书签导航;右侧界面主要用于显示 PDF 文档的内容。

### 10.3.2 菜单栏的实现

菜单栏的实现比较简单,这里笔者只设计了一个菜单项,即“文件”菜单项,并且为该菜单项添加了 4 个子菜单项,分别为“打开”、“打印”、“页面设置”和“退出”。菜单栏这部分在程序中的应用如图 10.13 所示。



图 10.13 菜单栏的实现

创建菜单栏的代码封装在一个名为 `CreateMenuBar()` 的方法中,该方法没有参数,返回值为 `JMenuBar` 对象,该方法的关键代码如下:

```
public JMenuBar CreateMenuBar() {
    final JMenuBar menuBar = new JMenuBar(); // 实例化一个 JMenuBar 对象
    menuBar.setBounds(0, 0, 1000, 25); // 设置该菜单栏的具体位置和大小
    final JMenu newItemMenuItem1 = new JMenu(); // 初始化一个菜单项
    newItemMenuItem1.setText("文件(F)"); // 为第一个菜单项设置名称
    menuBar.add(newItemMenuItem1); // 将“文件”菜单项添加到菜单栏中
    final JMenuItem newItemMenuItem_1 = new JMenuItem(); // 初始化一个子菜单项
    newItemMenuItem_1.setText("打开"); // 为子菜单项设置名称
    newItemMenuItem1.add(newItemMenuItem_1); // 将子菜单项添加到菜单项上
    newItemMenuItem_1.setToolTipText("Open PDF file"); // 为子菜单项设置提示文字
    newItemMenuItem_1.addActionListener(new java.awt.event.ActionListener() { // 为子菜单项添加事件监听器
        public void actionPerformed(ActionEvent evt) {
            doOpen(); // 调用打开 PDF 文档方法
        }
    });
    final JMenuItem newItemMenuItem = new JMenuItem(); // 初始化第二个子菜单项
    newItemMenuItem.setText("页面设置"); // 为菜单项设置名称
    // 将“页面设置”菜单项添加到菜单栏中
    newItemMenuItem1.add(newItemMenuItem);
    newItemMenuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            doPageSetup(); // 调用页面设置方法
        }
    });
    final JMenuItem newItemMenuItem_3 = new JMenuItem(); // 初始化第 3 个子菜单项
    newItemMenuItem_3.setText("打印"); // 为子菜单项设置名称
}
```





```

newItemMenuItem_3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent arg0) {
        doPrint(); // 调用页面打印功能
    }
});
newItemMenuItem1.add(newItemMenuItem_3); // 将子菜单项添加到菜单项中
final JMenuItem newItemMenuItem_2 = new JMenuItem(); // 实例化第3个子菜单项
newItemMenuItem_2.setText("退出"); // 为子菜单项设置名称
newItemMenuItem_2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent arg0) {
        // 在“退出”菜单项的事件监听器中调用 doClose()方法
        doClose();
        System.exit(0);
    }
});
newItemMenuItem1.add(newItemMenuItem_2); // 将子菜单项添加到菜单项中
return menuBar; // 返回该菜单项
}

```

doClose()方法用来将所有正在使用的对象置空，对程序起到一个很好的善后作用，它的关键代码如下：

```

public void doClose() {
    if(jp!=null){
        jpmain.removeAll(); // 如果 PDF 文档面板不为空，删除主面板上的所有组件
    }
    if (thumbs != null) {
        thumbs.stop(); // 关闭缩位图面板
    }
    thumbs = new ThumbPanel(null); // 将缩位图面板置空
    thumbscroll.getViewport().setView(thumbs);
    pdfFile = null;
}

```

### 10.3.3 工具栏的实现

工具栏的实现也比较简单，在这里笔者将该部分内容封装在一个名称为 CreateToolBar()方法的方法中，该方法返回值为 JToolBar 对象。在工具栏中，分别设置了“首页”、“上一页”、“下一页”、“尾页”、“页码定位文本框”、“最大化”、“最小化”、“全屏”等功能按钮，并为各个按钮配备了相应的事件监听器，这些内部都放置在 CreateToolBar()方法中。CreateToolBar()方法的关键代码如下：

```

private JToolBar CreateToolBar() { // 创建工具栏的方法
    JToolBar toolBar = new JToolBar(); // 实例化一个 JToolBar 对象
    toolBar.setBounds(0, 23, 1000, 40); // 设置该工具栏的位置与大小
    toolBar.setFloatable(false); // 设置该工具栏不可以移动
    // 为该工具栏设置一个边界
    toolBar.setBorder(new BevelBorder(BevelBorder.RAISED));
    firstButton = new JButton(firstAction);
    .....// 省略部分代码
    toolBar.add(firstButton); // 将“首页”按钮添加到工具栏上
    .....// 省略部分代码
    nextButton = new JButton(nextAction);
    toolBar.add(nextButton); // 将“下一页”按钮添加到工具栏上
    backButton = new JButton(prevAction);
    .....// 省略部分代码
}

```







```

toolBar.add(backButton); // 将“上一页”按钮添加到工具栏上
lastButton = new JButton(lastAction);
.....// 省略部分代码
toolBar.add(lastButton); // 将“尾页”按钮添加到工具栏上
pageField = new JTextField("-", 3);
.....// 省略部分代码
toolBar.add(pageField); // 将页码定位的文本框添加到工具栏上
jl = new JLabel();
// 将显示当前页码与总页码的标签添加到工具栏上
toolBar.add(jl);
toolBar.addSeparator(); // 在工具栏上添加分隔符
bigButton = new JButton(BigAction);
toolBar.add(bigButton); // 将“放大”按钮放置在工具栏上
.....// 省略部分代码
smallButton = new JButton(SmallAction);
.....// 省略部分代码
toolBar.add(smallButton); // 将“缩小”按钮放置在工具栏上
toolBar.addSeparator();
fullScreenButton=new JButton(fullScreenAction);
.....// 省略部分代码
toolBar.add(fullScreenButton); // 将“全屏”按钮放置在工具栏上
return toolBar; // 返回工具栏
}

```

### 10.3.4 左侧索引面板的实现

左侧索引面板主要包括缩位图导航及书签导航功能, 在程序中考虑到界面的美观, 将缩位图导航面板与书签导航面板放置在 JTabbedPane 组件中, 选项卡面板是一种大家比较熟悉的用户界面设置, 可以将一个复杂的面板分割成相关的选项子集, 然后使用 addTab() 方法将子集添加到选项卡面板中。

左侧索引面板的功能实现关键代码如下:

```

tabbedPane=new JTabbedPane(); // 实例化一个 JTabbedPane 对象
// 将缩位图导航面板添加到选项卡面板中, 并设置选项卡的图标
tabbedPane.addTab(null, CreatecdIcon.add("02.gif"), thumbscroll);
// 将一个空面板添加到选项卡面板中
tabbedPane.addTab(null, CreatecdIcon.add("05.gif"), null);
tabbedPane.setTabPlacement(JTabbedPane.LEFT); // 设置选项卡面板的选项卡布局为左边
getContentPane().add(tabbedPane); // 将选项卡添加到容器中
tabbedPane.setBounds(0, 63, 270, 600); // 设置选项卡的位置与大小

```

### 10.3.5 右侧界面的实现

主窗体中的右侧界面主要用于显示 PDF 文档的内容, 实质上就是在主窗体的右方放置一个滚动面板, 然后将主面板放置在该滚动面板中, 由于 PDF Render 组件将每页 PDF 文档以面板的形式返回, 所以可以将每页 PDF 文档添加到主面板中, 为了使 PDF 文档纵向显示, 可以为主面板添加网格布局, 并设置该布局管理器为一列。

实现这部分内容的关键代码如下:

```

contentPanel = new JScrollPane(); // 实例化一个滚动面板
// 实例化一个 PagePanel 对象, 它是 PDF Render 组件中放置 PDF 文档的面板

```





```

jp=new PagePanel();
.....// 省略部分代码
jpmain.setLayout(new GridLayout(0, 1));           // 将主面板设置为 0 行 1 列的网格布局
contentPanel.setViewPortView(jpmain);             // 将主面板添加到滚动面板中
contentPanel.setBounds(270, 63, 600, 600);        // 设置滚动面板的位置与大小
getContentPane().add(contentPanel);               // 将滚动面板添加到容器中

```

## 10.4 打开 PDF 文档

### 10.4.1 功能概述

打开 PDF 文档是使用 JFileChooser 控件显示文件选择器，在文件选择器中选择需要打开的 PDF 文档，并在窗体中显示所选 PDF 文档的内容，操作方法是选择“文件”|“打开”菜单项，在弹出的文件选择器中选择需要打开的 PDF 文档，然后单击“打开”按钮，将在窗体中显示所选择的 PDF 文档内容，效果如图 10.14 所示。

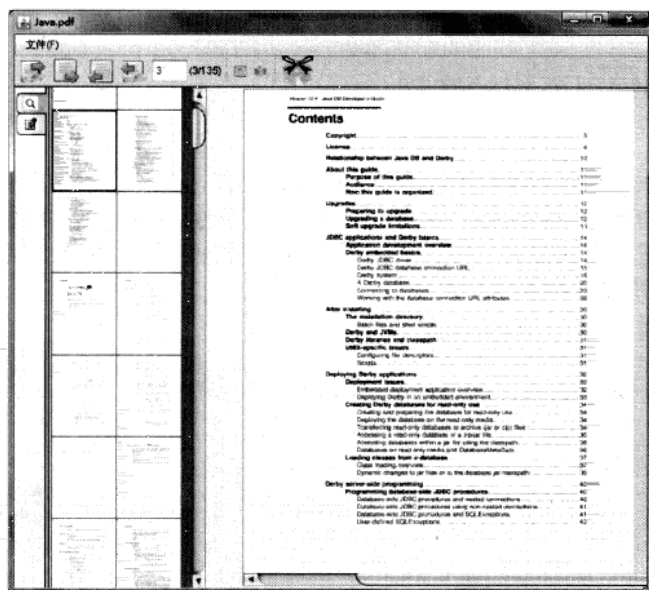


图 10-14 显示打开的 PDF 文档



### 10.4.2 创建文件选择器

288



当用户在菜单上选择“文件”、“打开”选项时，将在主窗体中弹出一个文件选择对话框，这个对话框映射了硬盘上的所有文件，用户可以在这个对话框中选择需要阅读的 PDF 文档。实现这个文件选择器使用 Swing 中的 JFileChooser 组件。在程序中将创建文件选择器的代码都封装在一个名称为 doOpen()的方法中，该方法用于实现打开 PDF 文档的一些





操作，它的关键代码如下：

```
public void doOpen() {
    try {
        JFileChooser fc = new JFileChooser();           // 创建 JFileChooser 实例
        fc.setCurrentDirectory(prevDirChoice);         // 设置当前路径
        fc.setFileFilter(pdfFilter);                   // 设置当前文件过滤器
        // 设置文件选择器，以允许选择多个文件
        fc.setMultiSelectionEnabled(false);
        // 弹出一个“Open File”文件选择器对话框
        int returnVal = fc.showOpenDialog(this);
        // 如果用户在“文件选择器”对话框中单击“确定”按钮
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            try {
                prevDirChoice = fc.getSelectedFile(); // 获取用户选择的路径
                RandomAccessFile raf = new RandomAccessFile(prevDirChoice
                    .getAbsolutePath(), "r"); // 实例化一个随机访问文件的实例
                FileChannel channel = raf.getChannel(); // 实例化一个 FileChannel 对象
                ByteBuffer buf = channel.map(FileChannel.MapMode.READ_ONLY,
                    0, channel.size());
                try {
                    pdfFile = new PDFFile(buf); // 实例化一个 PDFFile 实例
                } catch (IOException ioe) {
                    return;
                }
                .....// 省略部分代码
                for(int i=1;i<10;i++){
                    .....// 省略部分代码
                    PDFPage page = pdfFile.getPage(i);
                    PagePanel jp2 = new PagePanel();
                    .....// 省略部分代码
                    jpmain.add(jp2);
                    validate();
                    jp2.showPage(page);
                }
                .....// 省略部分代码
                backButton.setEnabled(true);           // 设定工具栏上的“上一页”按钮可用
                nextButton.setEnabled(true);            // 设定工具栏上的“下一页”按钮可用
                pageField.setEditable(true);            // 设定工具栏上的“页码”文本框可用
                // 设定工具栏上的标签显示当前与总页码
                j1.setText("("+(curpage+2)+"/"+pdfFile.getNumPages()+")");
                bigButton.setEnabled(true);              // 设定工具栏上的“放大”按钮可用
                smallButton.setEnabled(true);            // 设定工具栏上的“缩小”按钮可用
                lastButton.setEnabled(true);              // 设定工具栏上的“尾页”按钮可用
                fristButton.setEnabled(true);            // 设定工具栏上的“首页”按钮可用
                fullScreenButton.setEnabled(true);       // 设定工具栏上的“全屏”按钮可用
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



在 JFileChooser 组件使用 setCurrentDirectory() 方法时，可以将参数设置为指定目录，也可以将参数设置为 null，传入 null 会设置文件选择器指向用户的默认目录，此目录取决于操作系统，在 Windows 系统上通常是“我的文档”。





### 10.4.3 在文件选择器中只显示 PDF 文档

在文件选择器中只显示 PDF 文档的功能，在程序中使用比较普遍，它方便了用户查找指定类型的文件，这个功能在模块中的体现如图 10.15 所示。

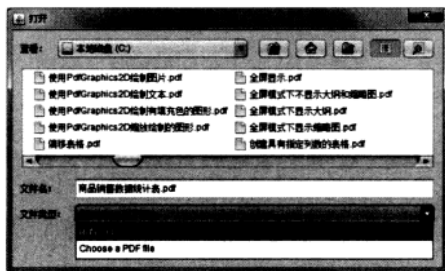


图 10.15 在文件选择器中只显示 PDF 文档

在图 10.15 中，选择“文件类型”下拉列表中的“所有文件”选项，文件选择器会将当前目录下的所有文件类型都列举出来，但是在没有选择的情况下，文件类型下拉列表默认会选择“choose a PDF file”选项，而在文件选择器中只会列举出 PDF 类型的文件。

在程序中为了实现这个功能可以使用 `setFileFilter()` 方法设置文件过滤器，这个方法的参数为 `FileFilter` 抽象类对象，使用它过滤显示文件集合，也就是在文件选择器中只为用户提供 PDF 类型的文件，在程序中为了获取一个 `FileFilter` 实例，定义一个继承 `FileFilter` 抽象类的匿名内部类，它返回一个 `FileFilter` 对象，它的关键代码如下：

```
FileFilter pdfFilter = new FileFilter() {
    public boolean accept(File f) {
        return f.isDirectory() || f.getName().endsWith(".pdf");
    }
    public String getDescription() {
        return "Choose a PDF file";
    }
};
```

当获取 `FileFilter` 实例后，就可以将其作为 `setFileFilter()` 方法的参数，这样在程序中定义的文件选择器就具有只显示 PDF 文档的功能。

### 10.4.4 使窗体标题栏显示 PDF 文档名称

在用户选定 PDF 文档后，本模块会自动将选定文档的名称作为主窗体标题栏上的标题。在模块中实现该功能如图 10.16 所示。

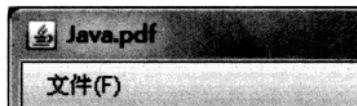


图 10.16 以 PDF 文档名称设置窗体标题栏







要想实现该功能需要获取用户选择的文件名称, 然后使用 setTitle() 方法进行设置, 实现该功能的关键代码如下:

```
docName = prevDirChoice.getName(); // 获取用户选择的 PDF 文件名称
setTitle(docName); // 设置窗体标题栏
```

### 10.4.5 显示 PDF 文档内容

在主面板上显示 PDF 文档内容主要是将每页 PDF 文档放置在布局为网格布局的主面板上, 由于该主面板的网格布局设置为 1 列, 所以每页 PDF 文档将纵向显示在主面板上。

获取每页 PDF 文档使用 PDF Render 组件中的 PDFFile 类, 该类中具有一个 getPage() 方法, 可以使用这个方法进行获取, 其中, getPage() 方法的参数表示 PDF 文档中的页码, 该方法的返回值为 PDFPage 类型对象, 然后使用 PagePanel 对象调用 showPage() 方法就可以实现某页的 PDF 文档的显示, 最后将该 PagePanel 对象添加到 JPanel 组件中即可。

实现这部分内容的关键代码如下:

```
for(int i=1;i<10;i++){
    contentPanel.setViewportView(jpmain); // 将主面板放置在滚动面板上
    .....// 省略部分代码
    PDFPage page = pdfFile.getPage(i); // 获取每页 PDF 文档
    PagePanel jp2 = new PagePanel(); // 实例化一个 PagePanel 对象
    jpmain.add(jp2); // 将该 PagePanel 对象添加到主面板中
    validate(); // 验证该容器中的所有子组件
    jp2.showPage(page); // 显示该页 PDF 文档
}
```



由于一次将所有 PDF 文档页面显示出来会耗费巨大内存, 所以这里笔者只将其中的前 10 页文档显示在主窗体上, 有兴趣的读者可以将上述代码中的 for 循环的终止值修改为 pdfFile.getNumPages(), 这样将显示 PDF 文档中的所有页面。

## 10.5 缩位图导航

### 10.5.1 功能概述

缩位图导航是文档页面的微型预览, 用户可以使用页面缩位图快速跳至选定的页面, 从而极大地方便了用户对 PDF 文档内容的查看, 缩位图导航的显示效果如图 10.17 所示。



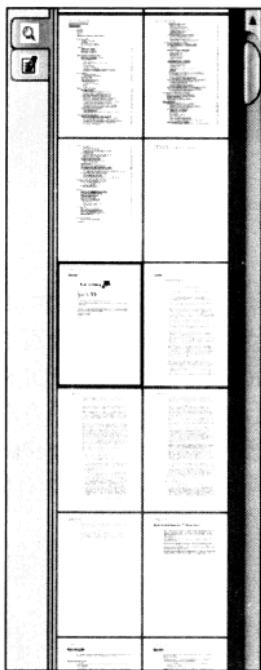


图 10.17 缩位图导航

### 10.5.2 实现缩位图面板

实现缩位图面板主要使用 PDF Render 中的 ThumbPanel 对象, 然后使用 getViewport() 方法将该对象添加到滚动面板中, 为了将该滚动面板放置在选项卡面板中, 可以使用 setComponentAt() 方法进行设置。

实现这部分功能的关键代码如下:

```
thumbs = new ThumbPanel(pdfFile); // 初始化一个 ThumbPanel 对象
thumbs.addPageChangeListener(this); // 为缩位图添加页面更换事件监听器
thumbscroll.getViewport().setView(thumbs); // 将缩位图面板添加到滚动面板中
thumbscroll.getViewport().setBackground(Color.gray); // 设置滚动面板的背景颜色
// 将上述滚动面板添加到选项卡面板中, 参数 0 代表第一个选项卡
tabbedPane.setComponentAt(0, thumbscroll);
```

### 10.5.3 实现缩位图索引功能



缩位图的索引功能是通过为 ThumbPanel 对象添加 PageChangeListener 事件监听器而实现的, 由于 MainFrame.java 类实现了 PageChangeListener 接口, 所以 thumbscroll 滚动条使用 addPageChangeListener() 方法时的参数为 this, 也就是 MainFrame 类的实例, 实现 PageChangeListener 接口的类需要重写 gotoPage() 方法, 在关键技术中笔者已经讲解过该方法, 它用于完成进入指定页面的功能, 当 ThumbPanel 对象添加这个监听事件后就具有





进入指定页面的功能。这部分功能实现的步骤如下。

- (1) 使 MainFrame.java 类实现 PageChangeListener 接口。
- (2) 重写 gotoPage() 方法。该方法关键代码如下：

```
public void gotoPage(int pagenum) {
    if (pagenum < 0) {
        pagenum = 0;
    } else if (pagenum >= pdffile.getNumPages()) {
        pagenum = pdffile.getNumPages() - 1;
    }
    forceGotoPage(pagenum);
}

public void forceGotoPage(int pagenum) {
    if (pagenum <= 0) {
        pagenum = 0;
    } else if (pagenum >= pdffile.getNumPages()) {
        pagenum = pdffile.getNumPages() - 1;
    }
    curpage = pagenum;
    // 更新在菜单栏上的页码定位文本框
    pageField.setText(String.valueOf(curpage + 1));
    // 更新在菜单栏上的页码显示标签
    jl.setText("(" + (curpage + 1) + "/" + pdffile.getNumPages() + ")");
    // 获取指定页面的 PDFPage 对象
    PDFPage pg = pdffile.getPage(pagenum + 1);
    if (jpmain != null) {
        jpmain.removeAll();
    }
    jpmain.add(jp);
    validate();
    if (fspp != null) {
        fspp.showPage(pg);
        fspp.requestFocus();
    } else {
        jp.showPage(pg);
        jp.requestFocus();
    }
    // 在窗体左侧缩位图面板中指出当前阅读的 PDF 页面，并且使用高亮边界将正在阅读的页面圈上
    thumbs.pageShown(pagenum);
}
```

- (3) 为该 ThumbPanel 对象添加 PageChangeListener 监听，关键代码如下：

```
thumbs = new ThumbPanel(pdffile);
thumbs.addPageChangeListener(this);
```

## 10.6 书签导航

### 10.6.1 功能概述

书签导航是文档页面的结构预览，用户可以使用书签快速跳至选定的页面，尤其是当文档的页码较多时，可以极大地方便用户对 PDF 文档的查阅，书签导航的显示效果如图 10.18 所示。



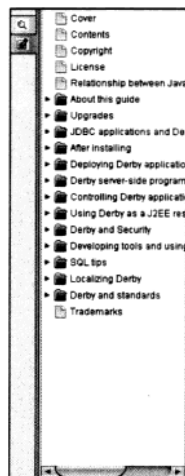


图 10.18 书签导航

## 10.6.2 实现书签面板

实现书签面板功能主要使用 `OutlineNode` 对象，该对象用于获取 PDF 文档的大纲，使用 `PDFFile` 对象调用 `getOutline()` 方法可以获得 `OutlineNode` 对象，在获取的同时需要捕捉 `IOException` 异常。

当获取到 `OutlineNode` 对象后，使用该对象初始化一个 `JTree` 实例，然后将树放置在滚动面板中，最后将滚动面板放置在选项卡面板中。编写书签面板的关键代码如下：

```
try {
    outline = pdfFile.getOutline();           // 获取 OutlineNode 对象
} catch (IOException ioe) {
}
if (outline != null) {
    if (outline.getChildCount() > 0) {        // 如果 OutlineNode 对象不为空
        JTree jt = new JTree(outline);        // 如果该节点的子节点数大于 0
        jt.setRootVisible(false);            // 创建 JTree 对象
        // 为该树添加 TreeSelectionListener 事件监听器
        jt.addTreeSelectionListener(this);
        JScrollPane jsp = new JScrollPane(jt); // 将树放置在滚动面板中
        tabbedPane.setComponentAt(1, jsp);    // 将滚动面板放置在选项卡面板中
    }
}
```



## 10.6.3 实现书签索引功能

若想实现书签索引功能，需要以 `OutlineNode` 对象为节点生成树，并为该树添加 `TreeSelectionListener` 事件监听器，由于 `MainFrame.java` 实现了 `TreeSelectionListener` 接口，所以 `JTree` 对象调用 `addTreeSelectionListener()` 方法的参数为 `this`。在 `MainFrame.java` 中重





写 TreeSelectionListener 接口中的 valueChanged() 方法, 这个方法的关键代码如下:

```
public void valueChanged(TreeSelectionEvent e) {
    // 如果已将第一个路径元素添加到选择中, 返回 true
    if (e.isAddedPath()) {
        // 返回第一路径元素中的最后一个组件
        OutlineNode node = (OutlineNode) e.getPath().getLastPathComponent();
        if (node == null) {
            // 如果这个值为空
            return; // 返回
        }
        try {
            // 返回此节点的用户对象, 并转型为 PDFAction 对象
            PDFAction action = node.getAction();
            if (action == null) {
                // 如果这个 Action 为空
                return; // 返回
            }
            if (action instanceof GoToAction) {
                // 获取 PDFDestination 对象
                PDFDestination dest = ((GoToAction) action).getDestination();
                if (dest == null) {
                    // 如果该对象为空
                    return; // 返回
                }
                PDFObject page = dest.getPage(); // 获取 PDFObject 对象
                if (page == null) {
                    // 如果该对象为空
                    return; // 返回
                }
                // 获取用户选择的页码
                int pageNum = pdfFile.getPageNumber(page);
                if (pageNum >= 0) {
                    // 如果该页码不小于 0
                    gotoPage(pageNum); // 调用 gotoPage() 方法获取指定的页面
                }
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

## 10.7 全屏显示 PDF 文档

### 10.7.1 功能概述

全屏显示 PDF 文档也为用户阅读 PDF 文档提供了方便条件, 当用户打开某个 PDF 文档, 并指定阅读某个页面, 单击工具栏上的“全屏”按钮时, 这个页面会以全屏的形式显示在屏幕上, 效果如图 10.19 所示, 当用户在全屏状态时按下键盘上的【Esc】键即可退出全屏状态, 除此之外, 本模块还为用户提供全屏状态下使用某些键控制页面翻页的功能。



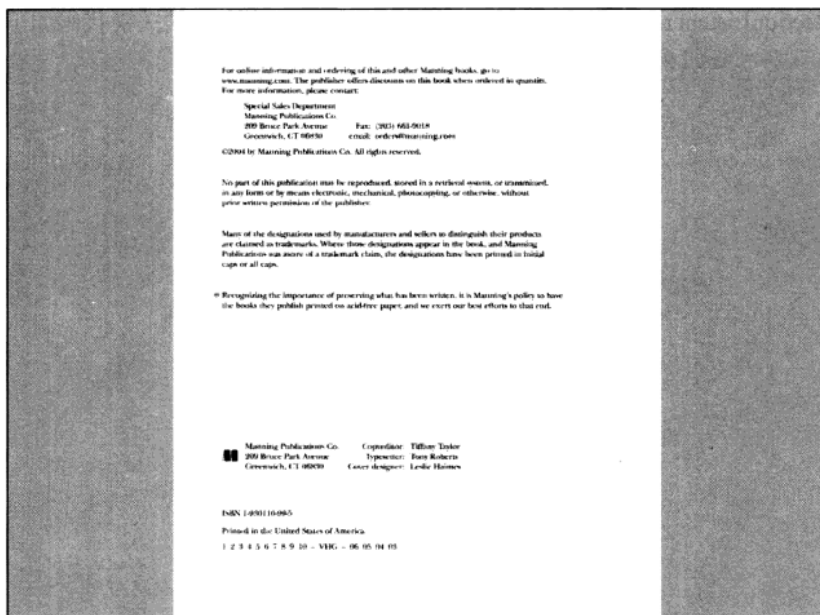


图 10.19 全屏显示 PDF 文档

### 10.7.2 在工具栏中添加“全屏”按钮

将“全屏”按钮添加到工具栏的具体实现过程如下。

- (1) 实例化一个 `JButton` 对象，用于表示“全屏”按钮。
- (2) 设置该按钮的相关属性。
- (3) 将该按钮添加到工具栏中。

将“全屏”按钮添加到工具栏的实现代码如下：

```
fullScreenButton=new JButton(fullScreenAction); // 初始化一个 JButton 按钮
fullScreenButton.setEnabled(false); // 设置该按钮不可用
fullScreenButton.setToolTipText("全屏"); // 设置该按钮的提示文字
// 设置该按钮在工具栏上的图标
fullScreenButton.setIcon(CreatecdIcon.add("fullScreen.gif"));
toolBar.add(fullScreenButton); // 将该按钮添加到工具栏上
```

### 10.7.3 实现全屏显示功能



使 `JButton` 按钮具有全屏显示功能主要是由 `Action` 对象来完成的，这个 `Action` 对象名称为 `fullScreenAction`，它继承了 `AbstractAction` 类，程序中以匿名内部类的形式实现该 `Action`，并重写 `actionPerformed()` 方法，实现该功能的关键代码如下：

```
Action fullScreenAction = new AbstractAction() {
    public void actionPerformed(ActionEvent evt) { //重写 actionPerformed() 方法
        // 调用 doFullScreen() 方法，该方法用于实现 PDF 文档全屏的功能
    }
}
```





```
doFullScreen((evt.getModifiers() & evt.SHIFT_MASK) != 0);
}
};
```

在上述代码中调用了 doFullScreen()方法, 该方法用于实现 PDF 文档全屏的功能, 该方法的参数是运算式(evt.getModifiers() & evt.SHIFT\_MASK) != 0 返回的结果, ActionEvent 类提供了 getModifiers()方法返回表示所有按下的修改键的整数值, 在程序中产生动作事件时, 用户可以按下【Alt】、【Ctrl】或【Shift】等键, 使用这个整数值与 ActionEvent 类成员常量 ALT\_MASK、CTRL\_MASK、META\_MASK 和 SHIFT\_MASK 进行位运算就可以在按钮或菜单项的事件处理程序中根据结果是否非零判断是否按下修改键, 在这里判断用户在单击“全屏”按钮的同时是否按下【Shift】键。

doFullScreen()方法实现了页面全屏的功能, 它的关键代码如下:

```
public void doFullScreen(boolean force) {
    setFullScreenMode(fullScreen == null, force); // 调用 setFullScreenMode()方法
}
```

在上述代码中可以看到, 事实上实现页面全屏功能的代码是 setFullScreenMode()方法, 该方法的参数为两个布尔型变量, 第一个参数获取该全屏面板是否为空, 而第二个参数获取用户在按下全屏按钮的同时是否按下【Shift】键, 该方法的关键代码如下:

```
public void setFullScreenMode(boolean full, boolean force) {
    if (full && fullScreen == null) { // 如果全屏面板为空
        fullScreenAction.setEnabled(false); // 将实现全屏的 Action 设置不可用
        new Thread(new PerformFullScreenMode(force)).start(); // 启动控制全屏的线程
        fullScreenButton.setSelected(true); // 设置“全屏”按钮被选择
    } else if (!full && fullScreen != null) { // 如果全屏面板不为空
        fullScreen.close(); // 关闭全屏面板
        fspp = null; // 全屏面板置空
        fullScreen = null;
        gotoPage(curpage); // 进入指定的页面
        fullScreenButton.setSelected(false); // 设置“全屏”按钮不被选择
    }
}
```

PerformFullScreenMode 类继承了 Runnable 接口, 在该类的 run()方法中实现页面全屏的功能, 该部分的关键代码如下:

```
class PerformFullScreenMode implements Runnable { // 类实现 Runnable 接口
    boolean force;
    public PerformFullScreenMode(boolean forcechoice) {
        force = forcechoice;
    }
    public void run() { // 重写 run()方法
        fspp = new PagePanel(); // 初始化全屏面板
        fspp.setBackground(Color.CYAN); // 设置全屏面板的颜色
        // 将主窗体显示 PDF 文档面板设置为空
        jp.showPage(null);
        //实例化 FullScreenWindow 实例
        fullScreen = new FullScreenWindow(fspp, force);
        fspp.addKeyListener(MainFrame.this); // 为全屏面板添加键盘事件
        gotoPage(curpage); // 进入指定页面
        fullScreenAction.setEnabled(true); // 设置全屏的 Action 可用
    }
}
```

由于 MainFrame.java 实现了 KeyListener 接口, 所以在为全屏面板添加 keyListener 事







件时方法的参数可以设置为 this,MainFrame.java 实现 KeyListener 接口需要重写 3 个方法,这 3 个方法分别为 keyPressed()、keyReleased() 及 keyTyped() 方法。

在这里笔者主要重写了 keyPressed() 与 keyTyped() 方法,在实现页面全屏的代码中为全屏面板添加了 keyListener 监听器, keyPressed() 方法描述的是键盘按下时触发的事件,在该方法中实现了键盘对全屏面板的控制。该方法的关键代码如下:

```
public void keyPressed(KeyEvent evt) {
    int code = evt.getKeyCode();
    if (code == evt.VK_LEFT) {
        doPrev();
    } else if (code == evt.VK_RIGHT) {
        doNext();
    } else if (code == evt.VK_UP) {
        doPrev();
    } else if (code == evt.VK_DOWN) {
        doNext();
    } else if (code == evt.VK_HOME) {
        doFirst();
    } else if (code == evt.VK_END) {
        doLast();
    } else if (code == evt.VK_PAGE_UP) {
        doPrev();
    } else if (code == evt.VK_PAGE_DOWN) {
        doNext();
    } else if (code == evt.VK_SPACE) {
        doNext();
    } else if (code == evt.VK_ESCAPE) {
        setFullScreenMode(false, false);
    }
}
```

// 获取键盘按下事件中的键的整数  
// 如果用户按下了键盘向左方向的键  
// 调用进入上一页方法  
// 如果用户按下了键盘向右方向的键  
// 调用进入下一页方法  
// 如果用户按下了键盘向上方向的键  
// 调用进入上一页方法  
// 如果用户按下了键盘向下方向的键  
// 调用进入下一页方法  
// 如果用户按下键盘【Home】键  
// 调用进入首页方法  
// 如果用户按下键盘【End】键  
// 调用进入尾页方法  
// 如果用户按下键盘【PageUp】键  
// 调用进入上一页方法  
// 如果用户按下键盘【PageDown】键  
// 调用进入下一页方法  
// 如果用户按下空格键  
// 调用进入下一页方法  
// 如果用户按下【Esc】键  
// 将退出全屏模式

通过为全屏面板添加 keyListener 监听事件,当用户进入全屏时可以使用键盘控制页面翻页。

keyType() 方法用于限制文本框输入的类型,在这里限制文本框只可以输入数字,在 MainFrame.java 类中重写该方法的关键代码如下:

```
public void keyTyped(KeyEvent evt) {
    char key = evt.getKeyChar();
    if (key >= '0' && key <= '9') {
        int val = key - '0';
        // pb 变量为 PageBuilder 类对象, PageBuilder 类为 MainFrame.java 的内部类
        pb.keyTyped(val);
    }
}
```

// 获取用户输入的字符  
// 如果用户输入的字符不在 0~9 之间  
// 将用户输入的字符与 0 字符进行相减运算

在上述代码中使用到 PageBuilder 类中的 keyTyped() 方法,该类的关键代码如下:

```
class PageBuilder implements Runnable {
    int value = 0;
    long timeout;
    Thread anim;
    static final long TIMEOUT = 500;
    public synchronized void keyTyped(int keyval) { // 定义的 keyTyped() 方法
        value = value * 10 + keyval; // 将用户输入字符转为相应数字
        timeout = System.currentTimeMillis() + TIMEOUT; // 设置线程停止时间
        if (anim == null) {
            anim = new Thread(this); // 创建线程实例
            anim.start(); // 开启线程
        }
    }
}
```







```
}  
}  
public void run() { // 重写 run() 方法  
    long now, then;  
    synchronized (this) {  
        now = System.currentTimeMillis(); // 获取当前时间  
        then = timeout; // 获取结束时间  
    }  
    while (now < then) { // 当当前时间小于结束时间时  
        try {  
            // 使线程休眠 500 毫秒, 其中 TIMEOUT 变量被赋予 500 毫秒  
            Thread.sleep(timeout - now);  
        } catch (InterruptedException ie) {  
        }  
        synchronized (this) {  
            now = System.currentTimeMillis();  
            then = timeout;  
        }  
    }  
    synchronized (this) {  
        gotoPage(value - 1); // 进入指定页面  
        anim = null; // 将线程置空  
        value = 0; // 将 value 变量置 0  
    }  
}  
}
```



# 第 11 章

---

## 动态考题模块

( Swing+MySQL 实现 )

无论是出试卷，改试卷还是统计分数，对考试管理者来说都是一项非常繁重的工作，但是随着信息技术的飞速发展，利用现代的信息技术可以使这项繁重的工作变得简单、快捷。一个完备的考试模块可以使学生及时地检测自己的学习效果，发现自己的不足，从而查漏补缺，更好地提高学习效率。考试模块中题目的生成、试卷的提交、成绩的批阅等都是自动完成的。只要形成一套成熟的题库就可以实现考试的自动化，省时省力。为了适应新形势的发展要求，笔者开发了动态考题模块，目的是支持学校进行考试、辅助考试管理、管理者管理考试、学生的上机练习等。通过本章的学习，读者能够学到：

- » 根据用户权限登录不同模块
- » 随机显示考题
- » 自动阅卷
- » 实现修改/删除指定试题
- » 显示倒计时





## 11.1 动态考题模块概述

### 11.1.1 模块概述

计算机技术没有应用到考试上时,组织一次考试至少要经过五步:人工出题、考生考试、人工阅卷、成绩评估和试卷分析,这是一项十分烦琐和非常容易出错的工作,教师的工作量非常大。很明显,传统的考试方式已经不再适应现代考试的需要。因此,需要创建动态考题模块来解决由于传统考试带来的不便。

根据传统考试的程序,在动态考题模块中设计了题目的生成、试卷的提交、成绩的批阅等功能。只要形成一套成熟的题库就可以实现考试自动化,这样一来,教师所要做的工作只是精心设计题目、维护题库,而不是组织考试,从而大大减轻了教师的负担。

### 11.1.2 功能结构

动态考题模块可以使用两种身份。一种是“考生”,另一种是“管理员”。考生登录模块后,可以进行考试、查分、修改密码等操作。管理员登录模块后,可以进行添加用户、修改用户信息、删除用户、添加考题、修改考题、删除考题等操作。动态考题模块的功能结构如图 11.1 所示。

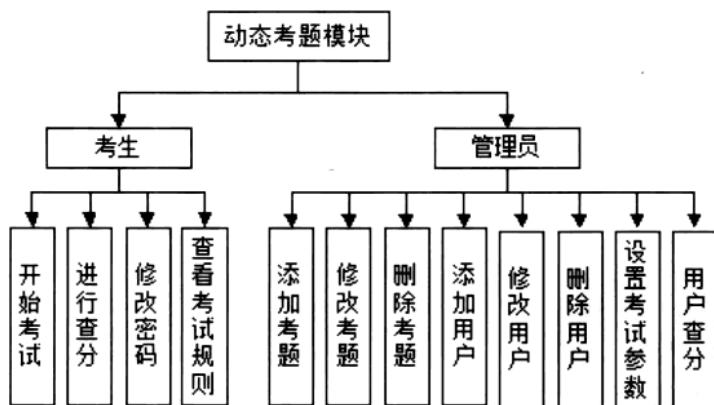


图 11.1 动态考题模块的功能结构图

### 11.1.3 程序预览

动态考题模块由多个窗体组成,下面仅列出几个典型界面,便于读者更好地了解程序。模块登录界面可以选择不同的用户类型进行登录,学生登录后进入考试界面,管理员登录





后进入模块管理界面。模块的登录界面如图 11.2 所示。

当用户以“管理员”身份登录模块后，即可进入模块的“后台管理”窗体。模块的“后台管理”窗体运行结果如图 11.3 所示。



图 11.2 登录窗体

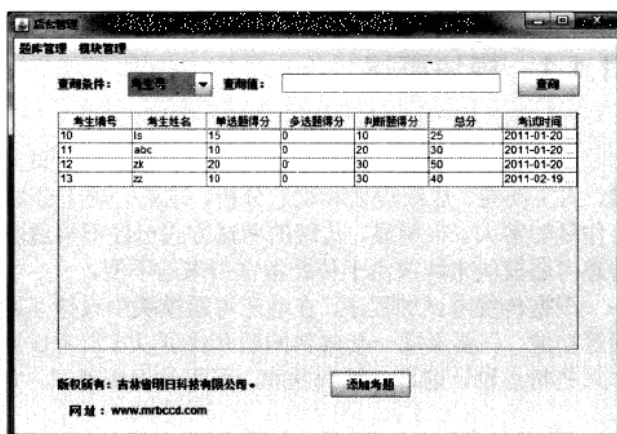


图 11.3 “后台管理”窗体

进入后台管理界面后，单击“模块管理”菜单下的“参数设置”菜单项，即可进入设置参数窗体界面，在此界面设置各题型的所占比例、设置每种题型、单道题的分数和考试总时间等。设置参数窗体界面如图 11.4 所示。

进入后台管理界面后，单击“模块管理”菜单下的“用户管理”菜单项，即可进入“用户管理”窗体，在此界面可以添加、修改和删除用户信息。“用户管理”窗体如图 11.5 所示。

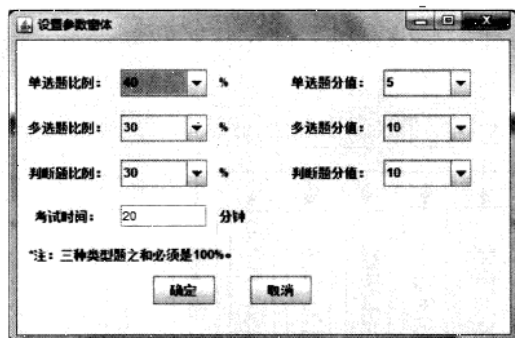


图 11.4 “设置参数”窗体



图 11.5 “用户管理”窗体



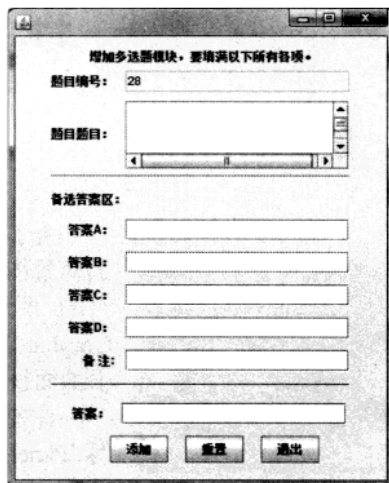
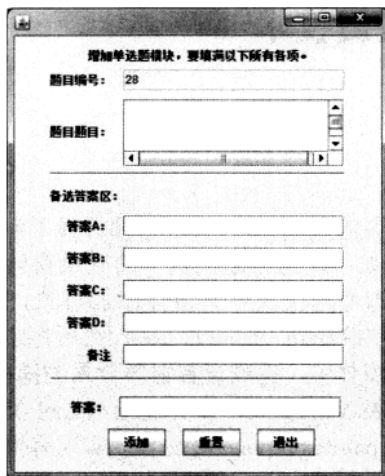
进入“后台管理”界面后，单击“题库管理”/“增加考题”/“增加单选题”菜单项，即可进入如图 11.6 所示的增加单选题界面，在此界面可以完成单选题的添加。



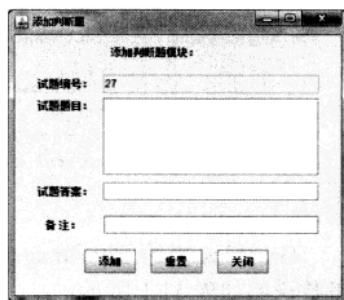
进入后台管理界面后，单击“题库管理”/“增加考题”/“增加多选题”菜单项，即可进入如图 11.7 所示的增加多选题界面，在此界面可以完成多选题的添加。



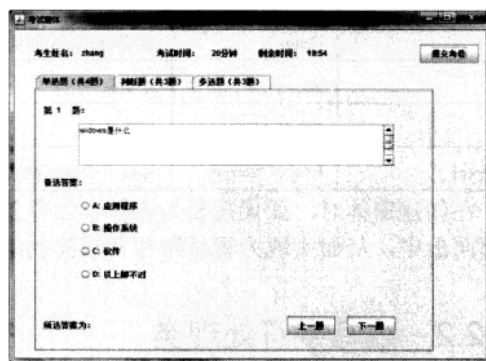




进入后台管理界面后,单击“题库管理”/“增加考题”/“增加判断题”菜单项,即可进入如图 11.8 所示的“添加判断题”窗体界面,在此界面可以完成判断题的添加。



进入“后台管理”界面后，单击“题库管理”/“修改/删除考题”菜单项，即可进入如图 11.9 所示的“修改/删除考题”窗体，在此界面可以完成考题的修改和删除操作。





## 11.2 关键技术

### 11.2.1 设置窗体背景

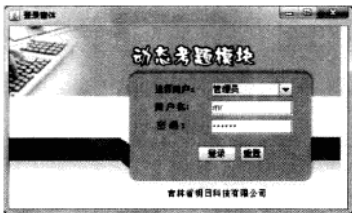


图 11.11 登录窗体

为了增强界面的美观性,很多程序都选择了为窗体增加背景图片。例如,动态考题模块中的登录窗体,如图 11.11 所示。动态考题模块中为窗体添加背景,是使用 Graphics 类将图片绘制到 JPanel 面板中,然后将绘制有图书的面板添加到窗体中,完成设置窗体背景的操作。

首先,创建类 MyJPanel,该类继承 JPanel 类,并重写 JPanel 类中的 paintComponent()方法,实现将指定的图片信息绘制到面板容器中。paintComponent()方法以 Graphics 类对象作为参数。MyJPanel 类的关键代码如下:

```
public class MyJPanel extends JPanel {
    private BufferedImage bfimage;
    public void paintComponent(Graphics g){
        try{
            URL url = getClass().getResource("/image/back.JPG"); //指定图片路径
            ImageIcon image = new ImageIcon(url); //创建 ImageIcon 对象
            g.drawImage(image.getImage(), 0, 0, this); //绘制图片
        }catch(Exception e){}
    }
}
```

Graphics 类的 drawImage()方法实现将图片绘制到指定组件中,该方法有 4 个参数,参数说明如表 11.1 所示。

表 11.1 drawImage()方法的参数说明

参数	类型	描述
img	Image	要绘制的指定图像。此参数如果为空,则此方法不执行任何操作
x	int	X 坐标
y	int	Y 坐标
observer	ImageObserver	转换了图像时要通知的对象

在创建窗体时,如果需要为窗体添加背景,可以将 MyJPanel 类的实例添加到窗体的内部面板中,从而实现为窗体添加背景的功能。

### 11.2.2 编写字符处理类

在与数据库交互过程中,可能会出现中文乱码问题。例如,从数据库中检索出来的是中文,在程序中可能会显示成乱码,影响程序的健全性。因此需要编写字符处理类,当向数据库中写数据、修改数据库中的数据,或从数据库中查数据等数据库操作时,进行字符





转码, 防止乱码情况发生。

编写 toChinese()方法, 用于将编码格式为 ISO8859\_1 的字符串, 转换为编码格式为 GBK 的字符串。toChinese()方法的代码如下:

```
public static String toChinese(String strvalue) {
    try {
        if (strvalue == null) {          //判断指定参数是否为空
            return "";
        } else {
            strvalue = new String(strvalue.getBytes("ISO8859_1"), "GBK")
                .trim();                //实现编码转化
            return strvalue;            //将转换后的字符串返回
        }
    } catch (Exception e) {
        return "";                      //有异常发生返回空值
    }
}
```

编写 toISO()方法, 实现将编码格式为 GBK 的字符串转换为编码格式为 “ISO8859\_1” 的字符串, toISO()方法的代码如下:

```
public static String toISO(String strValue) {
    try {
        if (strValue == null) {          //首先判断要进行转换的参数是否为空
            return "";
        } else {
            strValue = new String(strValue.getBytes("GBK"), "ISO8859_1")
                .trim();                //实现编码格式转化
            return strValue;            //将转换后的字符串返回
        }
    } catch (Exception e) {
        return "";                      //有异常发生返回空值
    }
}
```

### 11.2.3 编写获取时间方法

在程序设计中, 经常需要获取用户操作系统的当前时间。例如, 动态考题模块中获取用户设置考试参数的时间。本节将向读者介绍获取系统指定格式的日期时间所需要的重要类。

**类 SimpleDateFormat:** 可以选择任何用户定义的日期-时间格式的模式。定义的日期和时间格式由日期和时间模式字符串指定。例如, 要将日期-时间格式化为年-月-日格式, 可以使用 “yyyy-MM-dd” 模式来定义。

**Calendar 类:** Calendar 类是一个抽象类, 它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等日历字段之间的转换提供了一些方法, 并为操作日历字段(例如, 获得星期的日期)提供了方法。通过该类的 getTime()方法返回一个表示此 Calendar 时间值(从历元至现在的毫秒偏移量)的 Date 对象。

**Date 类:** 用于表示时间类。

下面是动态考题模块中进行日期时间格式化方法 getDateTime(), 该类的关键代码如下:





```
public static String getDateTime(){ //该方法返回值为 String 类型
    SimpleDateFormat format;
    //simpleDateFormat 类使得可以选择任何用户定义的时间-时间格式的模式
    Date date = null;
    Calendar myDate = Calendar.getInstance();
    //Calendar 的方法 getInstance, 以获得此类型的一个通用的对象
    myDate.setTime(new java.util.Date());
    //使用给定的 Date 设置此 Calendar 的时间
    date = myDate.getTime();
    //返回一个表示此 Calendar 时间值(从历元至现在的毫秒偏移量)的 Date 对象
    format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    //设置时间格式为年、月、日、时、分、秒
    String strRtn = format.format(date);
    //将给定的 Date 格式化为日期/时间字符串, 并将结果赋值给定的 String
    return strRtn;
}
```

## 11.2.4 编写 Java Bean

在程序设计中, 通常会将数据库中的数据表封装到一个 Java Bean 中。Java Bean 中包含的属性与数据库中表的字段相同, 并包含属性对应的 `getXXX()` 和 `setXXX()` 方法。在进行数据库操作时, 可操作相应的 Java Bean。例如, 题库表 `tb_question` 对应着 Java Bean 类 `Question`, 该类中的属性及属性对应的 `getXXX()`、`setXXX()` 方法代码如下:

```
public class Question {
    private int id; //对应 tb_question 数据表中的字段 id
    private int typeid; //对应 tb_question 数据表中的字段 typeid
    private String q_subject; //对应 tb_question 数据表中的字段 q_subject
    private String q_answer; //对应 tb_question 数据表中的字段 q_answer
    private String optionA; //对应 tb_question 数据表中的字段 optionA
    private String optionB; //对应 tb_question 数据表中的字段 optionB
    private String optionC; //对应 tb_question 数据表中的字段 optionC
    private String optionD; //对应 tb_question 数据表中的字段 optionD
    private String note; //对应 tb_question 数据表中的字段 note
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getNote() {
        return note;
    }
    public void setNote(String note) {
        this.note = note;
    }
    public String getOptionA() {
        return optionA;
    }
    public void setOptionA(String optionA) {
        this.optionA = optionA;
    }
    public String getOptionB() {
        return optionB;
    }
    public void setOptionB(String optionB) {

```







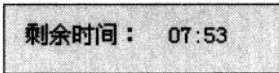
```

    this.optionB = optionB;
}
public String getOptionC() {
    return optionC;
}
public void setOptionC(String optionC) {
    this.optionC = optionC;
}
public String getOptionD() {
    return optionD;
}
public void setOptionD(String optionD) {
    this.optionD = optionD;
}
public String getQ_answer() {
    return q_answer;
}
public void setQ_answer(String q_answer) {
    this.q_answer = q_answer;
}
public String getQ_subject() {
    return q_subject;
}
public void setQ_subject(String q_subject) {
    this.q_subject = q_subject;
}
public int getTypeid() {
    return typeid;
}
public void setTypeid(int typeid) {
    this.typeid = typeid;
}
}

```

### 11.2.5 倒计时

在动态考题模块中,在考试界面设计了倒计时来提醒用户考试所剩余的时间,如图 11.12 所示。本模块在完成倒计时的显示是在单独的线程中完成的。当倒计时中显示“00:00”时,程序将强制考生交卷。



剩余时间： 07:53

图 11.12 倒计时

在完成显示倒计时,首先需要获取系统时间、管理员设置的考试时间。将当前系统时间与管理员设置的考试时间之和减去系统当前时间,将差作为标签的显示值添加到窗体中。通过 System 类中静态 currentTimeMillis()方法可获取系统当前时间,语法格式为:

```
System.currentTimeMillis()
```



该方法以 long 类型变量作为返回值,返回以毫秒为单位的当前时间。





若想将考试剩余时间显示成“分：秒”的形式，需要对时间进行格式化。动态考题模块中使用 `PrintWriter` 类的 `format()` 方法实现对时间的格式化，`format()` 方法语法格式为：

```
format(String format, Object... args)
```



`format`: 在格式字符串的语法中描述的格式字符串；`args` 为格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的数量是可变的，并且可以为零。

动态考题模块中完成倒计时的显示关键代码如下：

```
public void run() {
    int grade = 0;
    FindStat findStat = new FindStat();           //创建查询考试参数类对象
    Stat stat = new Stat();                       //创建持久化类对象
    stat.setId(1);
    Stat st = findStat.getStat(stat);             //获取考试参数方法
    int examtime = st.getExam_time();             //获取用户设置的考试时间
    long time = (long) (examtime * 60 * 1000L);   //获取考试时间的毫秒数
    Date date = new Date();
    StringWriter swer = new StringWriter();        //创建字符流对象
    StringBuffer sbf = swer.getBuffer();          //StringBuffer 类提供了字符串缓存区
    PrintWriter pw = new PrintWriter(swer);       //PrintWriter 类提供了文本输出流对象
    long cur = 0L;
    //获取系统当前时间与管理员设置的时间之和
    long end = System.currentTimeMillis() + time;
    while ((cur = end - System.currentTimeMillis()) > 0) {
        date.setTime(cur);
        pw.format("%1$tM:%1$tS", date);           //日期以“分：秒”形式进行格式化
        pw.flush();
        spareLabel.setText(sbf.toString());
        sbf.setLength(0);
        try {
            Thread.sleep(6L);
        } catch (InterruptedException e) {}
    }
    spareLabel.setText("00:00");                  //spareLabel 为 JLabel 对象，用于显示倒计时
    try {
        Thread.sleep(1200L);
    } catch (InterruptedException e) {}
    spareLabel.setText("时间到!!!");              //倒计时结束，强制交卷
    JOptionPane.showMessageDialog(this, "强制交卷!", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
    getGread();                                  //调用自定义计算考试成绩方法
}
```

## 11.3 登录窗体



### 11.3.1 功能概述

308



进入动态考题模块，必须经过登录窗体，根据不同的身份可以登录相应的界面。以考生身份登录模块的用户进入考生界面，进行考试。以管理员身份登录模块的用户进入管理界面，可以进行用户管理、考试参数管理、试题管理、查分等操作。另外，在学生进行登





录时,为了防止学生同时在一台计算机进行登录,或是重复进行考试,在登录时判断了当前登录账号是否已登录,或是当前登录账号是否登录过。登录窗体运行结果如图 11.13 所示。



图 11.13 登录窗体

### 11.3.2 编写验证用户是否合法的方法

登录窗体中要求用户输入合法的用户名、密码,因此需要编写按照用户名、密码检索用户的方法。为了保证用户信息的安全,在用户表中存储的用户密码采用了 MD5 加密技术。在判断用户输入的用户名和密码是否合法时,要将用户输入的密码进行加密后,与数据库中的密码进行对比,相同则表示用户输入的密码正确。MyMD5 类为将字符串进行加密的类,代码可参考光盘中源程序。验证用户是否合法的方法 getUser() 的完整代码如下:

```
/**
 * @param user 用户信息表 tb_use 对应的 Java Bean
 * @return user 对象
 */
public User getUser(User user) {
    String strSql = "select * from tb_use where UserName=? and PassWord =?";
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    MyMD5 myMd5 = new MyMD5();
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setString(1, ChDeal.toISO(user.getUserName()));
        pstmt.setString(2, myMd5.createPassWord(user.getPassWord()));
        System.out.println(myMd5.createPassWord(user.getPassWord()));
        rs = pstmt.executeQuery();
        while (rs.next()) {
            user.setId(rs.getInt("id"));
            user.setUserType(rs.getInt("UserType"));
            user.setUserName(rs.getString("UserName"));
            user.setPassWord(rs.getString("PassWord"));
            user.setHaveIn(rs.getInt("haveIn"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                rs.close();
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```





```
    }  
    return user;  
}
```

为了保证一个用户只能进行一次考试,在 tb\_use 表中设置了 havaIn 字段。如果 tb\_use 表中的 havaIn 字段值是“0”,表示该用户没有参加过考试;havaIn 字段值是“1”,表示该用户已经参加过考试。修改 tb\_use 表中 havaIn 字段值的方法代码如下:

```
/**  
 * @param userinfo 与用户表 tb_use 对应的 Java bean  
 * @return 成功修改 havaIn 值返回 true, 否则返回 false  
 */  
public boolean setUserHaveIn(User userinfo) {  
    boolean blnrec = true;  
    String strSql = "update tb_Use set havaIn = ?"; //修改 tb_Use 表的 SQL 语句  
    PreparedStatement pstmt = null;  
    try {  
        pstmt = conn.prepareStatement(strSql);  
        pstmt.setInt(1, userinfo.getHaveIn());  
        pstmt.executeUpdate(); //执行 SQL 语句  
    } catch (Exception e) {  
        e.printStackTrace();  
        blnrec = false;  
    } finally {  
        try {  
            if (pstmt != null) {  
                pstmt.close();  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    return blnrec;  
}
```

### 11.3.3 实现登录功能

当用户单击登录窗体中的“登录”按钮后,首先会判断用户是否输入“用户类型”、“用户名”、“密码”等。如果用户没有输入相应信息,就会给出提示信息,如图11.14所示。

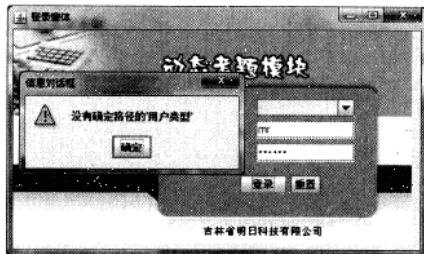


图 11.14 登录页面检测用户是否输入完整信息

“登录”按钮的单击事件中,首先完成检测用户是否输入“用户类型”、“用户名”、“密码”等信息,具体代码如下:





```
//判断用户是否使用“用户类型”
if (userChoicejComboBox.getSelectedIndex() == 0) {
    JOptionPane.showMessageDialog(this, "没有确定路径的'用户类型'",
        "信息对话框", JOptionPane.WARNING_MESSAGE);
    return; //退出程序
}
//判断用户是否输入“用户名”
if (UserNameTextField.getText().equals("")) {
    JOptionPane.showMessageDialog(this, "用户名不能为空",
        "密码不能为空", JOptionPane.WARNING_MESSAGE);
    return;
}
//判断用户是否输入“密码”
if (PassWordjTextField.getText().equals("")) {
    JOptionPane.showMessageDialog(this, "密码不能为空", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
    return;
}
```

如果用户将“用户类型”、“用户名”、“密码”输入完整后，将调用 `getUser()` 方法判断用户输入的数值是否合法。如果用户输入的用户名、密码正确，将根据用户身份，进入相应的模块，并将登录用户的 id 写入到 `save.txt` 文件中，具体代码如下：

```
private void enterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 省略了验证用户是否输入登录信息的代码
    MyFindUserDao findUser = new MyFindUserDao(); // 创建查询用户类对象
    User user = new User(); // 创建 tb_use 对象的 Java Bean 类对象
    user.setUserName(UserNameTextField.getText()); // 设置用户名
    user.setPassword(PassWordjTextField.getText()); // 设置密码
    User users = findUser.getUser(user); // 调用查询用户方法
    if ((users.getUserType() == 1 && !(userChoicejComboBox.getSelectedItem().
        equals("管理员")))) {
        JOptionPane.showMessageDialog(this, "登录身份不符", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if ((users.getUserType() == 0) && !(userChoicejComboBox.getSelectedItem().
        equals("考生")) {
        JOptionPane.showMessageDialog(this, "登录身份不符", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    // 登录身份为“考生”，进入后台管理页面
    if ((users.getId() != 0) && (users.getUserType() == 0)) {
        int id = users.getId();
        java.io.File file = new java.io.File("save.txt");
        try {
            if (file.exists()) {
                file.delete();
            }
            file.createNewFile();
            java.io.FileOutputStream out = new java.io.FileOutputStream(file);

            byte buy[] = (""+id).getBytes();
            out.write(buy); // 将登录用户写入 save.txt 文件中
            out.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 创建操作 tb_use 表类对象
    InsertUserDao insertUserDao = new InsertUserDao();
    insertUserDao.setUserHaveIn(users); // 调用修改数据表 tb_use 中的数据方法
    StudentExam studentExam = new StudentExam(); // 考生主页
    studentExam.setVisible(true);
    studentExam.setBounds(200, 200, 400, 300);
    studentExam.setTitle("考试窗体");
}
```





```
this.dispose();
}
//登录身份为“管理员”，进入后台管理页面
if((users.getId()!=0)&&(users.getUserType()==1)){
    dispose();
    ControllerFrame controller = new ControllerFrame();
    controller.setVisible(true);
    controller.setBounds(100,100, 700, 500);
}
else if(users.getId() == 0){
    JOptionPane.showMessageDialog(this,"用户名或密码错误","信息对话框",
    JOptionPane.WARNING_MESSAGE);
}
}
```

## 11.4 考试主窗体

### 11.4.1 功能概述

以考生身份进入模块后，会进入考试窗体主界面，如图 11.15 所示。在考试窗体主界面中，可进行考试、查分、修改密码等操作。

如果考生没有参加过考试，可以单击考试窗体中“考试”/“开始考试”菜单项，进入考试窗体开始参加考试，该窗体集合了试卷生成、试卷显示、时间显示、试卷提交、强制提交试卷及成绩的批阅等功能。用户进入动态考题模块后，将随机生成试题并添加到窗体中，以保证考试的公平性。窗体中试题的类型有三种，分别为单选题、多选题、判断题，不同类型的试题在不同的选项卡中显示。

在窗体的上方显示登录考试模块的考生姓名、考试的总时间、剩余时间等信息。考生答完试题后单击“提交”按钮提交试卷，将自动判卷并显示考试成绩。如果到达规定的考试时间考生仍未提交试卷，则程序将强制提交。考试窗体的运行效果如图 11.16 所示。



图 11.15 考试窗体主界面

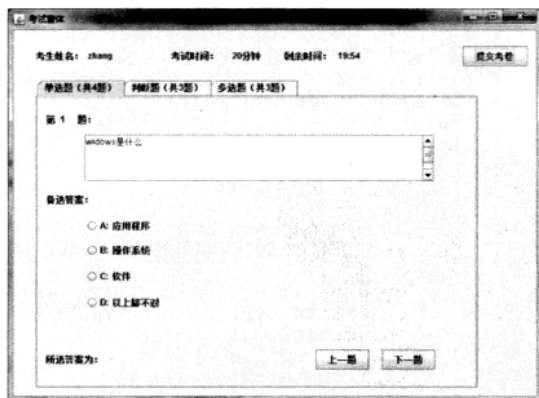


图 11.16 考试窗体运行结果





## 11.4.2 显示考生姓名

在用户登录窗体中完成将登录用户的 id 号保存在文本文件中。在考试窗体中通过考生的 id 号, 将考生姓名检索出来, 显示在考试窗体中。首先编写按 id 号检索用户方法。

```
/**
 * @param user 与 tb_use 表对应的 Java Bean User 类对象
 * @return User 类对象
 */
public User getUserID(User user) {
    String strSql = "select * from tb_use where id=?"; //按 id 号检索用户的 SQL 语句
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, user.getId()); //设置参数值
        rs = pstmt.executeQuery(); //执行 SQL 语句
        while (rs.next()) { //遍历查询结果集
            user.setId(rs.getInt("id"));
            user.setUserType(rs.getInt("UserType"));
            user.setUserName(rs.getString("UserName"));
            user.setPassword(rs.getString("PassWord"));
            user.setHaveIn(rs.getInt("havaIn"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                rs.close();
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return user;
}
```

在考试窗体中, 首先获取文本文件中的信息, 根据文件中的信息调用按 id 号检索用户的 getUserID 方法。并将检索出的考生姓名添加到考试窗体中, 具体代码如下:

```
java.io.File file = new java.io.File("save.txt"); //创建文件对象
java.io.FileInputStream in;
try {
    in = new java.io.FileInputStream(file);
    byte byt[] = new byte[1024];
    int len = in.read(byt); //读取文件信息
    String strid = new String(byt, 0, len);
    int id = Integer.parseInt(strid);
    MyFindUserDao findUser = new MyFindUserDao(); //创建封装查找用户类对象
    User user = new User(); //创建 Java Bean 类对象
    user.setId(id);
    User use = findUser.getUserID(user); //调用查找用户方法
    //jLabel1 为 JLabel 对象
    jLabel1.setText(ChDeal.toChinese(use.getUserName()));
} catch (Exception e) {
```







```
e.printStackTrace();  
}
```

### 11.4.3 显示考题

动态考题模块中的每套考题都是随机抽取的，再将随机抽取的试题显示在考试窗体中。在数据表 `tb_question` 中保存着所有考试题目数据。首先应编写检索某一类型试题集合的方法，具体代码如下：

```
/**  
 * @param tid 用于指定试题类型编号的 int 型变量  
 * @return 返回指定类型的试题集合  
 */  
public List findRadio(int tid){  
    String strSql = "select * from tb_question where typeid = '"+tid+"'";  
    PreparedStatement pstmt = null;  
    List list = new ArrayList(); //保存试题集合的 List 对象  
    ResultSet rs = null;  
    try {  
        pstmt = conn.prepareStatement(strSql);  
        rs = pstmt.executeQuery(); //执行 SQL 语句  
        while (rs.next()) {  
            int id = rs.getInt("id");  
            list.add(id); //向集合添加对象  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            if (pstmt != null) {  
                rs.close();  
                pstmt.close();  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    return list;  
}
```

获取指定试题类型的集合后，可以从指定试题集合中随机抽取一定数目的试题，保存在 `int` 类型数组中，具体代码如下：

```
/**  
 * @param countid 要随机抽取试题的个数  
 * @param typeName 试题类型名称  
 * @return int 型数组  
 */  
public int[] radowId(int countid,String typeName){  
    String SQL = "select * from tb_questionType where qName = '"+typeName+"'";  
    //根据试题名称查询试题类型表中的全部记录  
    PreparedStatement pstmt = null;  
    ResultSet rss = null;  
    int typeid = 0;  
    int[] idCount = null; //定义保存返回值的数组对象  
    try {  
        pstmt = conn.prepareStatement(SQL);
```







```

        rss = pstmt.executeQuery();           //指定 SQL 语句
        while(rss.next()){
            typeid = rss.getInt(1);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    List list = findRadio(typeid);             //调用获取指定试题类型集合的方法
    Set set = new HashSet();                  //创建 set 集合对象
    if(countid <= list.size()){
        for(int i = 0; i < list.size(); i++){
            int random = (int) (Math.random() * list.size()); //获取集合中的指定元素
            //添加到 set 集合中, 保证集合中的元素不为空
            set.add(Integer.parseInt(list.get(random).toString()));
        }
        if((set.size() == countid)){           //如果 set 集合中的元素与参数相等
            idCount = new int[countid];       //实例化数组对象
            Iterator it = set.iterator();      //遍历 set 集合
            int k = 0;
            while(it.hasNext()){
                //实例化数组中的每个元素
                idCount[k] = Integer.parseInt(it.next().toString());
                k++;
            }
        } else{                               //如果 set 集合元素的个数与参数不相同
            idCount = radowId(countid, typeName); //再次调用本方法
        }
        return idCount;
    }
    return null;
}

```

动态考题模块中显示的考试题个数、考试时间等内容, 都是从考试参数表中检索出来的。根据考试参数表中设置的各项套题的个数, 来决定考试窗体中各项试题的显示数目。因此需要编写检索考试参数表中信息的方法 `getStat`, 该方法以 `Stat` 对象作为参数, `Stat` 类是与考试参数表 `tb_stat` 对应的 Java Bean, 该类中的属性与 `tb_stat` 表中的字段相对应。方法 `getStat` 的具体代码如下:

```

/**
 * @param stat    该方法以与考试参数表 tb_stat 对应的 Java Bean 为 Stat 对象作为参数
 * @return        返回 Stat 对象
 */
public Stat getStat(Stat stat){
    String strSql = "select * from tb_stat where id=?";
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, stat.getId());
        rs = pstmt.executeQuery();
        while (rs.next()) { //将 stat 对象中属性设置成查询结果集中的数据信息
            stat.setId(rs.getInt("id"));
            stat.setJudge_BL(rs.getInt("judge_BL"));
            stat.setJudge_FS(rs.getInt("judge_FS"));
            stat.setExam_time(rs.getInt("exam_time"));
            stat.setMore_BL(rs.getInt("more_BL"));
        }
    }
}

```







```
        stat.setMore_FS(rs.getInt("more_FS"));
        stat.setRadio_BL(rs.getInt("radio_BL"));
        stat.setRadio_FS(rs.getInt("radio_FS"));
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pstmt != null) {
            rs.close();
            pstmt.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return stat;
}
```

考试窗体中使用了选项卡面板, 实现在同一个窗体中显示不同的考试题目。例如, 当考生单击“单选题”选项卡后, 将显示“单选题”试题, 如图 11.17 所示; 当考生单击“多选题”选项卡后, 将显示“多选题”试题, 如图 11.18 所示。

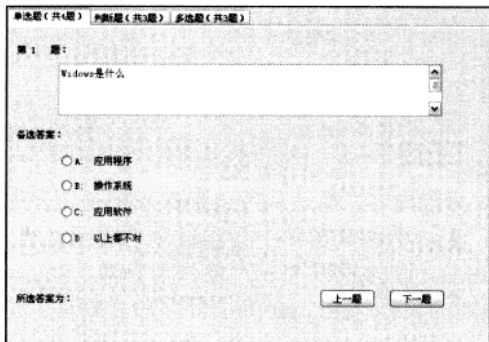


图 11.17 单选题试题

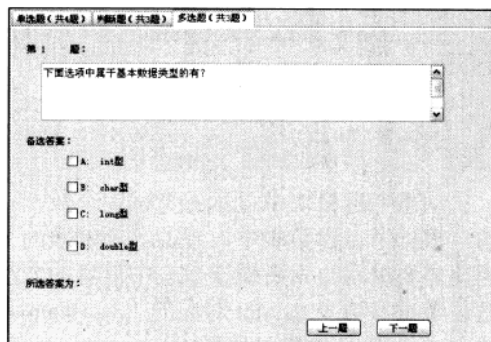


图 11.18 多选题试题

`javax.swing.JTabbedPane` 类实现了选项卡窗体。`JTabbedPane` 类中的 `addTab()` 方法可以给选项卡窗体设置标题、每个选项卡窗体显示的面板组件等。下面以显示单选题为例介绍动态考题模块在选项卡窗体中显示考题的实现过程。如图 11.18 所示, 考试题目是在 `JTextArea` 组件中显示的, 考题的备选答案是在 `JRadioButton` 组件中显示的。为了保证考生在解答单选题的过程中只允许选择一项, 需要将各个 `JRadioButton` 组件添加到一个按钮组里。首先创建所需要的各组件对象, 关键代码如下:

```
private javax.swing.JTabbedPane ExamTabbedPane;
private javax.swing.JPanel radiojPanel;
private javax.swing.JTextArea judgejTextArea;
private javax.swing.JTextArea radiojTextArea;
private javax.swing.JTextArea morejTextArea;
private javax.swing.JTextArea judgejTextArea;
private javax.swing.JRadioButton jRadioButton1;
private javax.swing.JRadioButton jRadioButton2;
private javax.swing.JRadioButton jRadioButton3;
```







```
private javax.swing.JRadioButton jButton4;  
private javax.swing.ButtonGroup buttonGroup1;
```

在考题窗体中的初始化方法 `initComponents()` 中, 实例化各组件, 并将从数据库中检索的考题信息作为各组件的显示信息。首先应该调用随机获取考题方法 `radowId()`, 由于该方法返回保存随机获取考试的试题编号 `int` 数组。当考试窗体第一次被初始化时, 获取数组中的第一个元素, 即试题的 `id` 号, 再根据 `id` 号检索试题, 并将检索出的信息显示在窗体中, 关键代码如下:

```
private void initComponents() {  
    buttonGroup1 = new javax.swing.ButtonGroup();  
    ExamTabbedPane = new javax.swing.JTabbedPane();  
    radiojPanel = new javax.swing.JPanel();  
    FindQuestionDao findQDao = new FindQuestionDao(); //创建检索考题信息类对象  
    FindStat findStat = new FindStat(); //创建获取考试参数类对象  
    Stat stat = new Stat(); //创建 Stat 类对象  
    stat.setId(1); //见说明  
    Stat st = findStat.getStat(stat); //查找考试参数  
    countid = findQDao.radowId(st.getRadio_BL()/10, ChDeal.toISO("单选题"));  
    //调用随机获取考题方法  
    Question question = new Question();  
    question.setId(countid[radio]);  
    Question ques = findQDao.getQuestion(question);  
    radiojTextArea.setColumns(20);  
    radiojTextArea.setRows(5);  
    //将考试题目显示在窗体中  
    radiojTextArea.setText(ChDeal.toChinese(ques.getQ_subject()));  
    jScrollPane1.setViewportView(radiojTextArea);  
    buttonGroup1.add(jButton1); //将单选按钮添加到按钮组里  
    buttonGroup1.add(jButton2);  
    buttonGroup1.add(jButton3);  
    buttonGroup1.add(jButton4);  
    //将考题答案显示在窗体中  
    jButton1.setText("A: "+ChDeal.toChinese(ques.getOptionA()));  
    answerjLabel.setText("备选答案: ");  
    jButton2.setText("B: "+ChDeal.toChinese(ques.getOptionB()));  
    jButton3.setText("C: "+ChDeal.toChinese(ques.getOptionC()));  
    jButton4.setText("D: "+ChDeal.toChinese(ques.getOptionD()));  
}
```



由于考试参数表 `tb_stat` 中只有一条记录, 该记录的 `id` 编号为“1”。因此可以通过指定 `id` 编号“1”来检索考试参数表中的内容。由于实现显示“多选题”、“判断题”与显示“单选题”方法类似, 这里不再给出, 请参考光盘中的源程序。

#### 11.4.4 转到上一题、下一题

当考生解答完本题后, 可以通过单击“下一题”按钮, 解答下一题, 在考试窗体中将显示下一题供考生解答。完成显示“下一题”考试题目, 要从保存随机抽取试题编号的 `int` 数组中获取“下一个”元素, 再根据该元素检索数据库, 将获取的信息作为窗体中各组件的显示信息, 关键代码如下:







```
private void downJButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    String count = countJLabel.getText(); //见代码说明  
    int countText = Integer.parseInt(count); //转换为 Integer 对象  
    //如果考生没有解答当前考题、给出提示信息  
    if((jRadioButton1.isSelected() == false)&&(jRadioButton2.isSelected() ==  
false)&&(jRadioButton3.isSelected()  
== false)&&(jRadioButton4.isSelected() == false)){  
        JOptionPane.showMessageDialog(this, "请回答完这道题后再执行该操作", "消息对话框", JOptionPane.WARNING_MESSAGE);  
        return; //退出方法  
    }  
    if(!count.equals(""+radioCount)){ //如果当前考题不是最后一题  
        int newCount = countText + 1; //获取保存随机抽取试题 int 数组  
        Question question = new Question();  
        question.setId(countid[countText]);  
        countJLabel.setText(""+newCount);  
        FindQuestionDao findQDao = new FindQuestionDao();  
        Question ques = findQDao.getQuestion(question);  
        //设置窗体中组件的显示信息  
        radioJTextArea.setText(ChDeal.toChinese(ques.getQ_subject()));  
        buttonGroup1.clearSelection(); //使单选按钮组中的按钮都不被选中  
        String strid = "";  
        if(radioList.size()>countText){ //radioList 为保存选题答案集合  
            for(int i = 0;i<radioList.size();i++){  
                strid = radioList.get(countText).toString(); //向集合添加值  
            }  
        }  
        if(!strid.equals("")&&(jRadioButton1.getText().substring(0, 1).  
equals(strid))){  
            jRadioButton1.setSelected(true);  
            radioanswer.setText("A");//radioanswer 为 JLabel 对象, 用于显示用户选择的答案  
        }  
        if(!strid.equals("")&&(jRadioButton2.getText().substring(0, 1).  
equals(strid))){  
            jRadioButton2.setSelected(true);  
            radioanswer.setText("B");  
        }  
        if(!strid.equals("")&&(jRadioButton3.getText().substring(0, 1).  
equals(strid))){  
            jRadioButton3.setSelected(true);  
            radioanswer.setText("C");  
        }  
        if(!strid.equals("")&&(jRadioButton4.getText().substring(0, 1).  
equals(strid))){  
            jRadioButton4.setSelected(true);  
            radioanswer.setText("D");  
        }  
        //设置窗体的显示信息  
        jRadioButton1.setText("A: "+ChDeal.toChinese(ques.getOptionA()));  
        jRadioButton2.setText("B: "+ChDeal.toChinese(ques.getOptionB()));  
        jRadioButton3.setText("C: "+ChDeal.toChinese(ques.getOptionC()));  
        jRadioButton4.setText("D: "+ChDeal.toChinese(ques.getOptionD()));  
    }  
    else{ //如果当前显示的考题是最后一题, 给出提示信息  
        JOptionPane.showMessageDialog(this, "已经是最后一题!", "信息对话框",  
JOptionPane.WARNING_MESSAGE);  
    }  
}
```

考试窗体中的选项卡标签中显示了当前考题共有几题, 并显示了当前考试是第几题。





在显示当前考题是第几题标签中显示的数值会随着当前考题而变化,如图 11.19 所示。当当前考题题号与选项卡窗体中显示的考题数相等后,提示用户已经是最后一题。

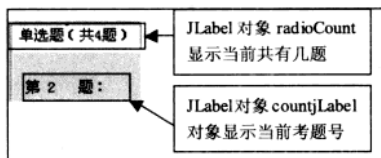


图 11.19 显示试题个数

当用户单击窗体中的“上一题”按钮后,程序会显示当前考题的上一道题。如果当前考题是第一题,将给出提示信息。

实现显示“上一题”,是从保存随机抽取试题的 int 数组中获取当前显示试题的前一个元素值,并用其值查询 tb\_question 表中的信息,将其显示在窗体组件中。“上一题”按钮的单击事件关键代码如下:

```
private void upjButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String count = countjLabel.getText(); //获取当前第几题标签的显示文本
    int countText = Integer.parseInt(count);
    if((jRadioButton1.isSelected() == false) && (jRadioButton2.isSelected() ==
        false) && (jRadioButton3.isSelected() == false) && (jRadioButton4.isSelected() ==
        false)){
        JOptionPane.showMessageDialog(this, "请回答完这道题后再执行该操作", "消息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if(!count.equals("1")){ //如果当前考题不是第一题
        int newCount = countText - 1;
        countjLabel.setText(newCount+"");
        Question question = new Question();
        question.setId(countid[newCount-1]); //获取保存随机抽取试题中的前一项元素
        FindQuestionDao findQuestionDao = new FindQuestionDao(); //创建检索考题类对象
        Question ques = findQuestionDao.getQuestion(question);
        radiojTextArea.setText(ChDeal.toChinese(ques.getQ_subject()));
        buttonGroup1.clearSelection(); //将单选按钮组中的对象清空
        String strid = "";
        if(radioList.size() > 0){
            for(int i = 0; i < radioList.size(); i++){
                strid = radioList.get(countText-2).toString(); //获取考生选择的答案
            }
        }
        if(!strid.equals("") && (jRadioButton1.getText().substring(0, 1).
            equals(strid))){
            jRadioButton1.setSelected(true); //如果考生选择“A”选项,将其设置为选中状态
            radioanswer.setText("A"); //答案标签显示为“A”
        }
        if(!strid.equals("") && (jRadioButton2.getText().substring(0, 1).
            equals(strid))){
            jRadioButton2.setSelected(true);
            radioanswer.setText("B");
        }
        if(!strid.equals("") && (jRadioButton3.getText().substring(0, 1).
            equals(strid))){
            jRadioButton3.setSelected(true);
            radioanswer.setText("C");
        }
    }
}
```





```

        if(!strid.equals("") && (jRadioButton4.getText().substring(0, 1).
            equals(strid))) {
            jRadioButton4.setSelected(true);
            radioanswer.setText("D");
        }
        //设置单选按钮的显示值
        jRadioButton1.setText("A: "+ChDeal.toChinese(ques.getOptionA()));
        jRadioButton2.setText("B: "+ChDeal.toChinese(ques.getOptionB()));
        jRadioButton3.setText("C: "+ChDeal.toChinese(ques.getOptionC()));
        jRadioButton4.setText("D: "+ChDeal.toChinese(ques.getOptionD()));
    }
    else{
        JOptionPane.showMessageDialog(this, "已经是第一题", "消息对话框",
            JOptionPane.WARNING_MESSAGE);
    }
}

```



由于“多选题”、“判断题”实现“上一题”、“下一题”的操作与“单选题”类似，这里不再给出，请参考光盘中的源程序。

### 11.4.5 实现自动阅卷

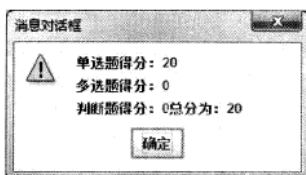


图 11.20 考生得分

getGread()具体代码如下:

```

private void getGread(){
    int radioGrage = 0;
    int examGrade = 0;
    int moreGrade = 0;
    Stat stat = new Stat(); //创建对应考试参数表 Java Bean Stat 对象
    stat.setId(1); //设置对象 id 号
    FindStat findStat = new FindStat(); //创建检索考试参数表对象
    Stat sta = findStat.getStat(stat); //查询指定的考试参数
    if(radioList.size()>0){ //如果保存单选题答案集合不为空
        for(int i = 0;i<radioList.size();i++){ //循环遍历集合
            int id = countid[i]; //获取保存随机抽取试题集合中的各元素
            FindQuestionDao findQuestionDao = new FindQuestionDao(); //创建查询考题类对象
            Question question = new Question(); //创建 Question 类对象
            question.setId(id);
            //查询试题方法
            Question question2 = findQuestionDao.getQuestion(question);
            if(question2.getQ_answer().equals(radioList.get(i))){

```







```

        radioGrage = radioGrage+sta.getRadio_FS();
    }
}
if(fullAnswer.size()>0){ //遍历保存多选题答案的集合
    for(int i = 0;i<fullAnswer.size();i++){
        int id = exitcount[i];
        Question ques = new Question();
        ques.setId(id);
        FindQuestionDao finddao = new FindQuestionDao();
        Question qetion = finddao.getQuestion(ques);
        String answer = qetion.getQ_answer();
        String newAnswer = answer.replace(","," ");
        if((qetion.getQ_answer().trim()).equals(fullAnswer.get(i).toString().trim())){
            moreGrade = moreGrade+sta.getMore_FS();
        }
    }
}
if(judgeList.size()>0){ //遍历保存判断题答案的集合
    for(int i = 0 ;i<judgeList.size();i++){
        int id = countEm[i];
        Question qtion = new Question();
        qtion.setId(id);
        FindQuestionDao findQ = new FindQuestionDao();
        Question qes = findQ.getQuestion(qtion);
        if(ChDeal.toChinese(qes.getQ_answer()).equals(judgeList.get(i).toString())){
            examGrade = examGrade+sta.getJudge_FS();
        }
    }
}
int sum = radioGrage + moreGrade + examGrade; //获取总分
Grade grade = new Grade();
File file = new File("save.txt");
try{
    //创建保存登录用户文本文件的对象
    FileInputStream in = new FileInputStream(file);
    byte byt[] = new byte[1024];
    int len = in.read(byt);
    String strbyte = new String(byt,0,len);
    int strid = Integer.parseInt(strbyte);
    User user = new User();
    user.setId(strid);
    MyFindUserDao findUserdao = new MyFindUserDao();
    //创建封装查找用户类 MyFindUserDao 对象
    User uid = findUserdao.getUserID(user);
    grade.setId(strid);
    grade.setUserName(uid.getUserName());
    grade.setRadioResult(radioGrage);
    grade.setFullResule(moreGrade);
    grade.setEsitResult(examGrade);
    grade.setBatsisResult(sum);
    grade.setDate(ChDeal.getDateTime());
    InsertGrade inserGrade = new InsertGrade(); //添加考分类对象
    inserGrade.setGradeDBbean(grade); //添加考分
    in.close();
    grade.setId(Integer.parseInt(strbyte));
}catch (Exception e) {
    e.printStackTrace();
}

```





```
}
JOptionPane.showMessageDialog(this, "单选题得分: "+radioGrage+"\n"+"多选题得分: "+moreGrade+"\n"+"判断题得分: "+examGrade+"总分为: "+sum, "消息对话框",
JOptionPane.WARNING_MESSAGE);
this.dispose();
}
```

## 11.5 管理员查分功能

### 11.5.1 功能概述

管理员具有最高的权限,主要任务是设定考试参数、管理题库、添加用户等。进入后台管理窗体后,管理员可通过考生号、考生姓名进行考分查询,如图 11.21 所示。

查询条件：		考生号	▼	查询值：		查询
考生编号	考生姓名	单选题得分	多选题得分	判断题得分	总分	考试时间
1	大熊猫	8	20	35	36	2008-09-12
4	阿阿	8	0	0	8	2008-09-11...

图 11.21 管理员查询考分窗体

### 11.5.2 按考生号查询成绩

当用户选择按“考生号”查询考分,并在“查询值”文本框中输入正确的查询条件时,程序将调用按“考生号”查询考分的方法,将查询结果显示在窗体中。首先编写按考生号查询考分的方法,具体代码如下:

```
/**
 * @param grade 对应考分表 Java Bean Grade 对象
 * @return      Grade 对象
 */
public Grade getGradeID(Grade grade){
    String strSql = "select * from tb_grade where id=?"; //定义查询考分表 SQL 语句
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, grade.getId());
        rs = pstmt.executeQuery(); //执行 SQL 语句
        while (rs.next()) { //循环遍历查询结果集
            grade.setId(rs.getInt("id"));
            grade.setUserName(rs.getString("userName"));
            grade.setRadioResult(rs.getInt("radioResult"));
            grade.setFullResule(rs.getInt("fullResule"));
            grade.setEsitResult(rs.getInt("esitResult"));
            grade.setBatsisResult(rs.getInt("batsisResult"));
            grade.setDate(rs.getString("date"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```







```

    } finally {
        try {
            if (pstmt != null) {
                rs.close();
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return grade;
}

```

在“查询”按钮的单击事件中,调用不同的查询方法,并将查询结果显示在窗体中,具体代码如下:

```

private void findButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //首先判断用户是否输入查询值
    if(nameTextField.getText().equals("")){
        JOptionPane.showMessageDialog(this, "请输入查询条件", "消息对话框",
            JOptionPane.WARNING_MESSAGE);
        return; //退出程序
    }
    //如果用户选择按“考生号”查询考分,程序将调用 getGradeID() 方法,并将查询结果显示在窗体中
    if(messageJchock.getSelectedItemAt().toString().equals("考生号")){
        FindGrade findGrade = new FindGrade();
        java.util.List<Grade> gradeList = findGrade.findGrade();
        if(!gradeList.isEmpty()){
            for(int i = 0;i<gradeList.size();i++){
                Grade grade = gradeList.get(i);
                try{
                    if(grade.getId() != Integer.parseInt(nameTextField.getText())){
                        }catch (Exception e) {
                            JOptionPane.showMessageDialog(this,
                                "必须输入数字!", "信息对话框", JOptionPane.WARNING_MESSAGE);
                        }
                    if(grade.getId() == Integer.parseInt(nameTextField.getText())){
                        Grade gradeId = findGrade.getGradeID(grade);
                        if(gradeId != null){
                            ((DefaultTableModel) jTable1.getModel()).setRowCount(0);
                            ((DefaultTableModel) jTable1.getModel()).addRow(new Object[] {
                                grade.getId(), ChDeal.toChinese(grade.getUser
                                    Name()), grade.getRadioResult(),
                                grade.getFullResule(), grade.getEsitResult(), grade.getBatsisResult(),
                                ChDeal.toChinese(grade.getDate())});
                        }
                        if(gradeId== null){
                            JOptionPane.showMessageDialog(this,
                                "没有要查询的考生成绩!", "信息对话框",
                                JOptionPane.WARNING_MESSAGE);
                        }
                    }
                }
            }
        }
        // ...省略了按考生姓名查询成绩的代码
    }
}

```

### 11.5.3 按考生姓名查询考分

当管理员选择按“考生姓名”查询考分后,程序将调用按“考生姓名”查询考分方法,将查询结果显示在窗体中。按“考生姓名”查询考分方法的代码如下:







```
/**
 * @param grade 与数据表 tb_stat 对应的 Java Bean Stat 对象
 * @return Stat 对象
 */
public Grade getGradeName(Grade grade){
    //按考生姓名检索数据表 SQL 语句
    String strSql = "select * from tb_grade where userName=?";
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, grade.getId());
        rs = pstmt.executeQuery(); //执行 SQL 语句
        while (rs.next()) { //遍历查询结果集
            grade.setId(rs.getInt("id"));
            grade.setUserName(rs.getString("userName"));
            grade.setRadioResult(rs.getInt("radioResult"));
            grade.setFullResule(rs.getInt("fullResule"));
            grade.setEsitResult(rs.getInt("esitResult"));
            grade.setBatsisResult(rs.getInt("batsisResult"));
            grade.setDate(rs.getString("date"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                rs.close();
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return grade;
}
```

在“查询”按钮的单击事件中，调用不同的查询方法，并将查询结果显示在窗体中，具体代码如下：

```
private void findButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //首先判断用户是否输入查询值
    if(nameTextField.getText().equals("")){
        JOptionPane.showMessageDialog(this, "请输入查询条件", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
        return; //退出程序
    }
    //...省略了按考生号查询成绩的代码
    //如果用户选择按“考生姓名”查询考分，程序将调用 getGradeName() 方法，并将查询结果显示在窗体中
    if(messageJchock.getSelectedItem().toString().equals("考生姓名")){
        FindGrade findGrade = new FindGrade();
        Grade grade = new Grade();
        grade.setUserName(nameTextField.getText());
        Grade grage = findGrade.getGradeName(grade);
        if(grage == null){
            JOptionPane.showMessageDialog(this,
            "没有要查询的考生成绩！", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        }
        else{
            ((DefaultTableModel) jTable1.getModel()).setRowCount(0);
            ((DefaultTableModel) jTable1.getModel()).addRow(new Object[] {
                grage.getId(), ChDeal.toChinese(grage.getUserName()),
                grage.getRadioResult(),
            });
        }
    }
}
```







```
grage.getFullResule(),grage.getEsitResult(),grage.
getBatsisResult(),
ChDeal.toChinese(grage.getDate())});
    }
}
```

## 11.6 添加考题

### 11.6.1 功能概述

当用户单击后台管理窗体中的“添加考题”按钮后,进入“添加试题”窗体,可实现添加考题。“添加考题”窗体如图 11.22 所示。

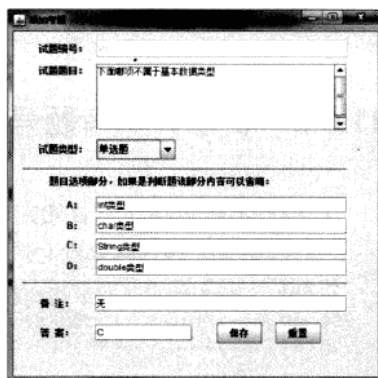


图 11.22 “添加考题”窗体

### 11.6.2 使用 List 集合存储所有考题

在如图 11.22 所示的“添加考题”窗体中,考题编号是以“只读”形式显示给用户的,不允许用户自行修改考题的编号。在“考题编号”文本框中显示的数据信息是根据试题表 tb\_question 中的记录总数进行加 1 得到的。首先编写获取数据表 tb\_question 中所有记录的方法 findQuestion(), 该方法以 List 集合作为返回值,具体代码如下:

```
public List findQuestion() {
    //查询数据表 tb_question 中的所有记录方法
    String strSql = "select * from tb_question";
    Statement pstmt = null;
    ResultSet rs = null;
    List lstList = new ArrayList();           //实例化 List 对象
    try {
        pstmt = conn.createStatement();
        rs = pstmt.executeQuery(strSql);      //执行 SQL 语句
        while (rs.next()) {                  //循环遍历查询结果集
            Question question = new Question();
            question.setId(rs.getInt("id"));
            question.setTypeid(rs.getInt("typeid"));
            question.setQ_subject(rs.getString("q_subject"));
            question.setQ_answer(rs.getString("q_answer"));
            question.setOptionA(rs.getString("optionA"));
            question.setOptionB(rs.getString("optionB"));
            question.setOptionC(rs.getString("optionC"));
            question.setOptionD(rs.getString("optionD"));
            question.setNote(rs.getString("note"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return lstList;
}
```





```
        lstList.add(question); //将集合添加信息
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pstmt != null) {
            rs.close();
            pstmt.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return lstList;
}
```

### 11.6.3 自动计算考题号

在添加考题窗体中，首先完成显示考题编号，具体代码如下：

```
testQuestionId.setText("试题编号: "); //testQuestionId 为 JLabel 对象
int id = 0;
//创建 FindQuestionDao 类对象
FindQuestionDao findQuestion = new FindQuestionDao();
java.util.List<Question> questList = findQuestion.findQuestion();
//调用检索数据表 tb_question 中全部数据的方法
if(questList.size()>0){
    for(int i = 0;i<questList.size();i++){
        Question question = questList.get(i);
        id = question.getId(); //获取数据表中最后一条记录的 id 编号
    }
}
int qid = id+1; //将 id 加 1
qidjTextField.setText(""+qid); //设置“考题编号”文本框中的显示信息
qidjTextField.setEnabled(false); //设置“考题编号”文本框为不可修改状态
```

### 11.6.4 保存考题

完成添加考题功能，首先应编写向保存考题数据表 tb\_question 中添加数据的方法，具体代码如下：

```
/**
 * @param question 与数据库 tb_question 对应的 java bean Question 对象
 * @return 当成功添加数据后，该方法返回 true，否则返回 false
 */
public boolean setQuestionDBbean(Question question) {
    boolean blnrec = true;
    String strSql = "insert into tb_question"
        + " values(?, ?, ?, ?, ?, ?, ?, ?)"; //添加试题 SQL 语句
    System.out.println(strSql);
    PreparedStatement pstmt = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, question.getId()); //设置参数
        pstmt.setInt(2, question.getTypeid());
        pstmt.setString(3, question.getQ_subject());
        pstmt.setString(4, question.getQ_answer());
        pstmt.setString(5, question.getOptionA());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return blnrec;
}
```







```

        pstmt.setString(6, question.getOptionB());
        pstmt.setString(7, question.getOptionC());
        pstmt.setString(8, question.getOptionD());
        pstmt.setString(9, question.getNote());
        pstmt.executeUpdate();
    } catch (Exception e) {
        e.printStackTrace();
        blnrec = false;
    } finally {
        try {
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return blnrec;
}

```

当用户添加完考题后,单击“添加考题”中的“保存”按钮后,程序会判断用户是否将考题信息添加完整,并调用添加试题方法 setQuestionDBbean(),完成考题的添加。“保存”按钮的单击事件的代码如下所示:

```

private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if(qSubjectjTextArea.getText().equals("")){ //判断“试题题目”是否为空
        JOptionPane.showMessageDialog(this, "请输入考试题目", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return; //退出方法
    }
    if(qanswerTextField.getText().equals("")){ //判断“试题答案”文本域是否为空
        JOptionPane.showMessageDialog(this, "请输入考题答案", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if((qTypejComboBox.getSelectedItem().toString().equals("多选题"))
        &&(qanswerTextField.getText().length()>0)
        &&(qanswerTextField.getText().indexOf(",")==-1)){
        JOptionPane.showMessageDialog(this, "多选题的答案要求格式如: 'A,B,C,D'", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    Question question = new Question(); //创建与考题表对应的 Java Bean 对象
    //设置 question 对象的 id 属性
    question.setId(Integer.parseInt(qidjTextField.getText()));
    FindQuestionDao insertQuestion = new FindQuestionDao();
    QuestionType questionType = new QuestionType();
    questionType.setQName(ChDeal.toISO(qTypejComboBox.getSelectedItem().toString()));
    QuestionType qtype = insertQuestion.getQuestionType(questionType);
    question.setTypeid(qtype.getId());
    question.setQ_subject(ChDeal.toISO(qSubjectjTextArea.getText()));
    question.setQ_answer(ChDeal.toISO(qanswerTextField.getText().trim()));
    question.setOptionA(ChDeal.toISO(optionATextField.getText()));
    question.setOptionB(ChDeal.toISO(optionBTextField.getText()));
    question.setOptionC(ChDeal.toISO(optionCjTextField.getText()));
    question.setOptionD(ChDeal.toISO(optionDTextField.getText()));
    question.setNote(ChDeal.toISO(noteField.getText()));
    insertQuestion.setQuestionDBbean(question);
    int n = JOptionPane.showConfirmDialog(this, "确定输入的数据正确吗?", "信息对话框",
        JOptionPane.YES_NO_OPTION);
    if(n == JOptionPane.YES_OPTION){
        JOptionPane.showMessageDialog(this, "考题添加成功", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
    }
}

```





## 11.7 修改/删除考题设计

### 11.7.1 功能概述

当用户单击“后台管理”窗体中的“题库管理”/“修改/删除考题”菜单项，将弹出如图 11.23 所示的考题列表，通过该列表可实现修改/删除考题。

试题编号	试题类型	试题题目	答案A	答案B	答案C	答案D	备注
1	1	windows	B	应用程序	操作系统	软件	以上都不对
2	1	下面哪个	A	SELECT	ad	ad	无
3	3	Java语言	正确				
4	1	下面哪个	A	String类型	int类型	char类型	boolean
5	2	WWW上	A.B.C.D	全球网	万维网	World Wi	以上叙述
6	3	数据库	错误				
7	1	下面哪个	D	注册服务器	配置本地	引入和导	为Windo
8	1	下面哪个	D	注册服务器	配置本地	引入和导	为Windo
9	1	下面哪个	D	注册服务器	配置本地	引入和导	为Windo
10	3	String	错误				
11	3	数组的下	错误				
12	3	SQL语言	正确				
13	2	Internet	A.B.C.D	E-mail	FTP	Telnet	WWW
14	3	defg	dfg				

图 11.23 “修改/删除考题”窗体

### 11.7.2 实现修改考题

图 11.24 “修改试题”窗体

当用户选择考题列表中的任一项时，单击“修改”按钮后，程序会将用户选择要进行修改的考题号写入名称为“tableQuestionId.txt”的文本文件中，再根据文件中信息检索数据表 tb\_question 获取相应信息，显示在如图 11.24 所示的“修改试题”窗体中，用户可根据其内容对试题进行修改。

在“修改试题”窗体中显示试题，应该从数据库中将对的数据检索出来，再将其值设置为窗体中组件的显示值，读者可参考光盘中的源程序。下面介绍完成修改试题的实现过程，首先编写修改某项试题的方法，具体代码如下：

```
/**
 * @param question 与试题表 tb_question 对应的 Java Bean 类 Question 对象
 * @return 成功修改将返回 true，否则返回 false
 */
public boolean updateQuestionDBbean(Question question) {
    boolean blnrec = true; //定义保存方法返回值对象
```



```
String strSql = "update tb_question set q_subject=?,q_answer=?,optionA=?,
optionB=? " +
",optionC=?,optionD=?, note=? where id=?"; //修改试题 SQL 语句
PreparedStatement pstmt = null;
try {
    pstmt = conn.prepareStatement(strSql);
    pstmt.setString(1, question.getQ_subject()); //设置语句参数
    pstmt.setString(2, question.getQ_answer());
    pstmt.setString(3, question.getOptionA());
    pstmt.setString(4, question.getOptionB());
    pstmt.setString(5, question.getOptionC());
    pstmt.setString(6, question.getOptionD());
    pstmt.setString(7, question.getNote());
    pstmt.setInt(8, question.getId());
    pstmt.executeUpdate(); //执行 SQL 语句
} catch (Exception e) {
    e.printStackTrace();
    blnrec = false;
} finally {
    try {
        if (pstmt != null) {
            pstmt.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return blnrec;
}
```

在修改试题窗体中显示考题编号的文本框笔者将其设置为“只读”形式，不允许用户修改。用户完成试题的修改工作后，单击“修改”按钮，程序将调用修改试题方法 `updateQuestionDBbean()`，完成试题的修改工作。在“修改”按钮的单击事件中，首先判断用户是否将相应的信息填写完整，具体代码如下：

```
private void updateJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //Question 为与数据表 tb_question 对应的 Java Bean
    Question question = new Question();
    if (subjectJTextArea.getText() == null) { //判断用户是否输入试题题目
        JOptionPane.showConfirmDialog(this, "请确定要修改的问题的题目！",
            "信息对话框", JOptionPane.WARNING_MESSAGE);
        return; //退出方法
    }
    if (answerJTextField.getText() == null) { //用户是否输入完整的试题答案
        JOptionPane.showConfirmDialog(this, "请确定要修改问题的答案！",
            "信息对话框", JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (index == 0) { //如果用户不是对判断题进行修改操作
        if ((optionA.getText() == null) || (optionBJText.getText() == null)
            || (optionJText.getText() == null) || (optionDjText.getText() ==
            null)) {
            JOptionPane.showConfirmDialog(this, "请将备选答案添加完整",
                "信息对话框", JOptionPane.WARNING_MESSAGE);
        }
        return;
    }
    //设置 Question 各项属性信息
    question.setId(Integer.parseInt(idJTextField.getText()));
    question.setQ_subject(ChDeal.toISO(subjectJTextArea.getText()));
    question.setQ_answer(ChDeal.toISO(answerJTextField.getText()));
    question.setOptionA(ChDeal.toISO(optionA.getText()));
    question.setOptionB(ChDeal.toISO(optionBJText.getText()));
    question.setOptionC(ChDeal.toISO(optionJText.getText()));
    question.setOptionD(ChDeal.toISO(optionDjText.getText()));
}
```





```
question.setNote(ChDeal.toISO(jTextArea2.getText()));
FindQuestionDao findQuestionDao = new FindQuestionDao();
//调用修改试题方法
boolean bool = findQuestionDao.updateQuestionDBbean(question);
if(bool == true){
    JOptionPane.showMessageDialog(this, "试题修改成功!", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
}
```

### 11.7.3 实现删除试题

当用户在“修改/删除考题”窗体中选择某一项试题，单击“删除”按钮后，程序会提示用户是否删除试题。在用户确定要删除试题时，完成试题的删除操作。首先编写删除指定试题方法，具体代码如下：

```
/**
 * @param lybId 要进行删除试题的 id 编号
 * @return 删除成功将返回 true，删除试题失败将返回 false
 */
public boolean delQuestion(int lybId) {
    boolean blnrec = true; //保存返回值的 boolean 对象
    String strSql = "delete from tb_question where id = ?"; //执行删除试题 SQL 语句
    PreparedStatement pstmt = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, lybId); //设置参数
        pstmt.executeUpdate(); //执行 SQL 语句
    } catch (Exception e) {
        blnrec = false;
        e.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return blnrec;
}
```

在“修改/删除”考题窗体的“删除”按钮的单击事件中，首先给出提示信息。在用户确定要删除试题时，调用删除试题方法 delQuestion()，删除试题，具体代码如下：

```
private void deleteButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if(tempCount == null){
        //tempCount 保存要删除的试题编号
        JOptionPane.showMessageDialog(this, "没有指定要删除的问题!", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return; //退出程序
    }
    else{
        int n = JOptionPane.showConfirmDialog(this, "确定要删除数据吗?", "确认对话框",
            JOptionPane.YES_NO_OPTION); //给出提示信息
        if(n == JOptionPane.YES_OPTION){
            FindQuestionDao findQuestion = new FindQuestionDao();
            //调用删除试题方法
            boolean bool = findQuestion.delQuestion(tempCount);
            if(bool == true){
                JOptionPane.showMessageDialog(this, "试题删除成功!", "信息对话框",
                    JOptionPane.WARNING_MESSAGE);
            }
        }
    }
}
```





```

    }
    if(n == JOptionPane.NO_OPTION){
        return;
    }
}
}

```

## 11.8 考试参数设置

### 11.8.1 功能概述

动态考题模块的后台管理中添加了设置考试参数模块,通过用户设置的考试参数可确定考试模块中每道题所占的比例、每道题的分值、考试总时间等信息。在设置时 3 种类型的试题比例相加之和要等于 100。设置参数窗体如图 11.25 所示。

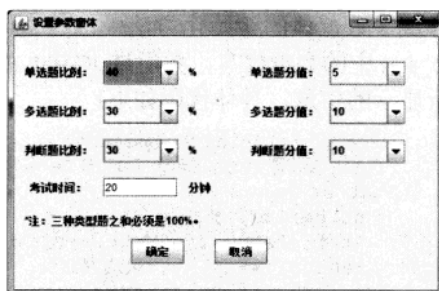


图 11.25 设置参数窗体

### 11.8.2 在下拉列表中显示内容

由于考试参数表中只保存一条记录,编号为 1,因此每次对考试参数进行设置时,就是对数据表 tb\_stat 中的数据进行修改。在设置考试参数窗体中,完成将数据表中的数据显示在窗体中。首先应编写检索数据表 tb\_stat 中数据的方法,具体代码如下:

```

/**
 * @param stat 该方法以与考试参数表 tb_stat 对应的 Java Bean 为 Stat 对象作为参数
 * @return 返回 Stat 对象
 */
public Stat getStat(Stat stat){
    String strSql = "select * from tb_stat where id=?"; //检索考试参数表 SQL 语句
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, stat.getId()); //设置参数
        rs = pstmt.executeQuery(); //执行 SQL 语句
        while (rs.next()) {
            stat.setId(rs.getInt("id")); //设置 Stat 对象的各项属性
            stat.setJudge_BL(rs.getInt("judge_BL"));
            stat.setJudge_FS(rs.getInt("judge_FS"));
            stat.setExam_time(rs.getInt("exam_time"));
            stat.setMore_BL(rs.getInt("more_BL"));
            stat.setMore_FS(rs.getInt("more_FS"));
            stat.setRadio_BL(rs.getInt("radio_BL"));
            stat.setRadio_FS(rs.getInt("radio_FS"));
        }
    }
}

```





```
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pstmt != null) {
            rs.close();
            pstmt.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return stat;
}
```

在设置考试参数窗体中,除考试时间外都采用了下拉列表显示各项考试参数。在设置考试参数窗体的初始化方法中,首先调用检索考试参数表的方法 `getStat()`,获取数据表 `tb_stat` 中的数据信息,并通过 `JComboBox` 类中的 `setSelectedItem()` 方法,设置下拉列表中所显示的信息,具体代码如下:

```
private void initComponents() {
    //创建与数据表 tb_stat 对应的 Java Bean 类 Stat 对象
    Stat st = new Stat();
    st.setId(1); //设置变量的 id 编号
    Stat stat = findStat.getStat(st); //调用检索考试参数方法
    radioBL.setText("单选题比例:"); //radioBL 为 JLabel 对象
    radiojComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] {
        "80", "70", "60", "50", "40", "30", "20", "10" })); //设置下拉列表中的数据模型
    //设置“单选题比例”下拉列表的选中选项
    radiojComboBox.setSelectedItem(""+stat.getRadio_BL());
    messageJLabel.setText("%");
    moreBL1.setText("多选题比例:");
    morejComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] {
        "80", "70", "60", "50", "40", "30", "20", "10" }));
    //设置“单选题比例”下拉列表的选中选项
    morejComboBox.setSelectedItem(""+stat.getMore_BL());
    jLabel2.setText("%");
    judgeBL.setText("判断题比例:");
    judgeJComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] {
        "80", "70", "60", "50", "40", "30", "20", "10" }));
    //设置“判断题比例”下拉列表的选中选项
    judgeJComboBox.setSelectedItem(""+stat.getJudge_BL());
    signJLabel.setText("%");
    examTime.setText(" 考试时间:");
    //设置“考试时间”文本框显示值
    examTimejTextField.setText(""+stat.getExam_time());
    minuteJLabel.setText("分钟");
    radiojLabel.setText("单选题分值:");
    radioMarkjComboBox.setModel(new javax.swing.DefaultComboBoxModel(new
        String[] { "1", "5", "8", "10" }));
    //设置“单选题分值”下拉列表选中选项
    radioMarkjComboBox.setSelectedItem(""+stat.getRadio_FS());
    morejLabel.setText("多选题分值:");
    moreMarkjComboBox.setModel(new javax.swing.DefaultComboBoxModel(new
        String[] { "1", "5", "8", "10" }));
    //设置“多选题分值”下拉列表选中选项
    moreMarkjComboBox.setSelectedItem(""+stat.getMore_FS());
    judgejLabel.setText("判断题分值:");
    judgeMarkjComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new
        String[] { "1", "5", "8", "10" }));
    //设置“判断题分值”下拉列表选中项
}
```







```
judgeMarkjComboBox1.setSelectedItem(""+stat.getJudge_FS());
}
```

### 11.8.3 实现考试参数设置

下面编写修改数据表 tb\_stat 中数据信息的方法 updateStatDBbean(), 该方法的参数是与数据表 tb\_stat 对应的类 Stat 的实例, 返回值为 boolean 类型, 具体代码如下:

```
/**
 * @param stat    与数据表 tb_stat 对应 Java Bean 类 Stat 对象
 * @return        当数据修改成功返回 true, 否则返回 false
 */
public boolean updateStatDBbean(Stat stat) {
    boolean blnrec = true;           //保存返回值的 boolean 对象
    String strSql = "update tb_stat set radio_BL=?,more_BL=?,judge_BL=?,radio_FS=? "
        + ",more_FS=?,judge_FS=?, exam_time=? where id=?";
    System.out.println(strSql);
    PreparedStatement pstmt = null;
    try {
        pstmt = conn.prepareStatement(strSql);
        pstmt.setInt(1, stat.getRadio_BL()); //设置 SQL 语句中的参数值
        pstmt.setInt(2, stat.getMore_BL());
        pstmt.setInt(3, stat.getJudge_BL());
        pstmt.setInt(4, stat.getRadio_FS());
        pstmt.setInt(5, stat.getMore_FS());
        pstmt.setInt(6, stat.getJudge_FS());
        pstmt.setInt(7, stat.getExam_time());
        pstmt.setInt(8, stat.getId());
        pstmt.executeUpdate();
    } catch (Exception e) {
        e.printStackTrace();
        blnrec = false;
    } finally {
        try {
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return blnrec;
}
```

当用户单击设置参数窗体中的“确定”按钮后, 将调用修改考试参数表的 updateStatDBbean()方法, 完成考试参数的修改, 具体代码如下:

```
private void okjButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String radioBox = radiojComboBox.getSelectedItemAt().toString();
    //radiojComboBox 为 JComboBox 对象, 单选题比例下拉列表
    String moreBox = morejComboBox.getSelectedItemAt().toString();
    //morejComboBox 为 JComboBox 对象, 多选题比例下拉列表
    String judgeBox = judgeJComboBox.getSelectedItemAt().toString();
    //judgeJComboBox 为 JComboBox 对象, 判断题比例下拉列表
    if((Integer.parseInt(radioBox)+
        Integer.parseInt(moreBox)+Integer.parseInt(judgeBox))!=100){
```





```
JOptionPane.showMessageDialog(this, "题目比例之和必须等于 100!", "信息对话框",
JOptionPane.WARNING_MESSAGE);
return;
}
if(examTimejTextField.getText() == null){ //判断“考试时间”文本框是否为空
    JOptionPane.showMessageDialog(this, "没有设置考试时间!",
        "信息对话框", JOptionPane.WARNING_MESSAGE);
}
Stat stat = new Stat(); //创建 Stat 对象
stat.setId(1);
stat.setJudge_BL(Integer.parseInt(judgeBox)); //设置 Stat 对象的各项参数
stat.setJudge_FS(Integer.parseInt(judgeMarkjComboBox1.getSelectedItem().toString()));
stat.setMore_BL(Integer.parseInt(moreBox));
stat.setExam_time(Integer.parseInt(examTimejTextField.getText()));
stat.setMore_FS(Integer.parseInt(moreMarkjComboBox.getSelectedItem().toString()));
stat.setRadio_BL(Integer.parseInt(radioBox));
stat.setRadio_FS(Integer.parseInt(radioMarkjComboBox.getSelectedItem().toString()));
FindStat findSta = new FindStat();
boolean bool = findSta.updateStatDBbean(stat); //调用修改考试参数方法
if(bool == true){ //考试参数修改成功, 给出提示信息
    JOptionPane.showMessageDialog(this, "考试参数设置成功!", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
}
}
```





# 第 12 章

---

## 多功能查询模块

( Swing+SQL Server 2005 实现 )

在开发应用程序时，对数据的查询是必不可少的，每开发一个程序都要单独制作一个查询窗体，其查询窗体的界面设置与技术基本相同，为了节省不必要的工作时间，可以根据查询窗体的特性，制作一个多功能查询模块，该模块可以接收主程序传过来的 SQL 语句，对单数据表及组合表添加查询条件，并将正确的查询条件返回给调用程序。通过本章的学习，读者能够学到：

- » SELECT 查询语句的使用
- » 多条件查询
- » 获取数据库中字段的描述信息
- » 检测用户是否填写正确的查询条件
- » 在项目中添加第三方类库



## 12.1 多功能查询模块概述

### 12.1.1 模块概述

多功能查询模块是一个用于拼接 SQL 语句的模块，需要对 saveSQL.txt 配置文件事先进行配置，它本身无法单独使用。saveSQL.txt 配置文件可以进行手工配置，也可以编写程序对它进行操作，配置 saveSQL.txt 完成后，多功能查询模块就可以读取 saveSQL.txt 中的信息，根据 saveSQL.txt 中读取的数据库名称与 SQL 语句，把 SQL 语句显示在多功能查询模块的窗体中。同时用户可以通过多功能查询模块的界面操作 SQL 语句。根据多功能查询模块的特点，总结出本模块应该实现以下功能：

- ☐ 通过输入/输出流实现数据互传。
- ☐ 可以与任何 Java 程序进行连接。
- ☐ 为了方便用户操作，在多功能查询模块中需要获取数据表中英文字段的描述信息。
- ☐ 可以向 SQL 语句中添加查询条件。
- ☐ 只能对 SQL 语句的条件部分进行修改。
- ☐ 支持模糊查询、多条件查询。

### 12.1.2 业务流程

制作多功能查询模块，首先应该了解通过输入/输出流，获取数据应该注意的几项内容，其次需要掌握 SQL 语句的数据查询结构。在指定查询条件时，由于查询数据表中的字段可能会是英文字段名，需要根据数据库中设置的数据表的描述信息来为用户建立查询条件。在返回用户设置的查询条件时，需要根据查询条件中的字段的描述信息，检索数据表中的字段信息，再将用户添加的查询条件返回给主程序。如图 12.1 所示为多功能查询模块的业务流程图。

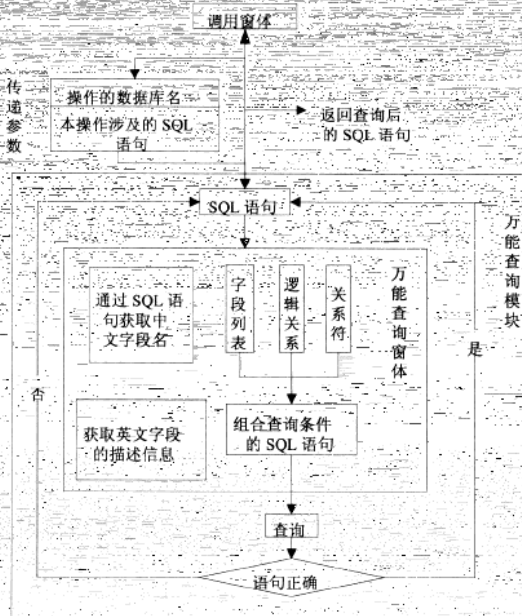


图 12.1 多功能查询业务流程图





### 12.1.3 程序预览

使用多功能查询模块, 需要有一个主程序来调用该模块。例如, 笔者设计了一个主程序, 在主程序中显示的是数据表中的大量数据信息, 通过该窗体的“查看”按钮可以调用多功能查询模块, 运行结果如图 12.2 所示。

在查询窗体中, 可根据窗体中的各种查询条件、数据表字段、逻辑关系等设置各种查询语句, 多功能查询模块的窗体运行结果如图 12.3 所示。

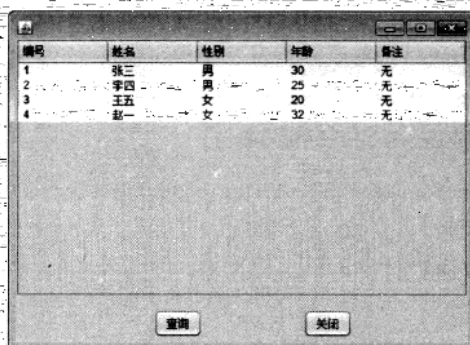


图 12.2 调用多功能查询模块窗体

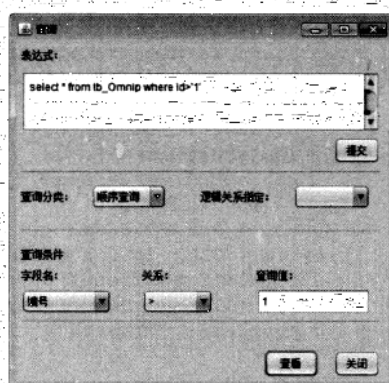


图 12.3 多功能查询模块主窗体

## 12.2 关键技术

### 12.2.1 JDBC 技术

JDBC 用来将 Java 程序与数据库进行连接, 在遵循 Java 语句规则的同时, 可以使用标准的 SQL 语句访问任何数据库。本模块中各个功能的实现都离不开 JDBC 技术, 例如, 获取数据库中指定数据表的字段的描述信息等。需要注意的是 JDBC 技术访问数据库, 并不能直接访问数据库, 必须依赖于数据库厂商提供的 JDBC 驱动程序。多功能查询模块是基于 SQL Server 2005 数据库开发的应用程序。要使 Java 应用程序与 SQL Server 2005 数据库建立联系, 首先要在项目中加载连接 SQL Server 2005 所需的驱动程序。

JDBC 技术主要完成以下几个任务。

- ☐ 与数据库建立连接。
- ☐ 向数据库发送 SQL 语句。
- ☐ 处理从数据库返回的结果。

通常, 使用 JDBC 技术操作数据库需要按照以下步骤。

#### 1. 连接数据库

要访问数据库, 首先应该建立与数据库的连接。建立与数据库的连接, 一般可分为以





下几个步骤。

(1) 通过 `java.lang` 包的静态方法 `forName()` 来加载 JDBC 驱动程序。如果加载失败将抛出 `ClassNotFoundException` 异常。

(2) 通过 `java.sql` 包中的 `DriverManager` 的静态方法 `getConnection()` 建立数据库连接。该方法中的 3 个参数分别用于指定连接数据库的路径、用户名和密码。

下面是多功能查询模块中与数据库建立连接的代码，数据库名称从 `MyConnection.getDBName()` 方法中获取：

```
public Connection getConnection() {
    try {
        // 加载数据库驱动
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        StringBuilder stringBuilder = new StringBuilder();
        // 设置连接数据库 URL
        stringBuilder.append("jdbc:microsoft:sqlserver://localhost:1433;" +
            "DatabaseName=");
        stringBuilder.append(MyConnection.getDBName());
        // 获取数据库连接
        con = DriverManager.getConnection(stringBuilder.toString(), "sa", "");
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return con;
}
```

### 2. 向数据库发送 SQL 语句

获取与数据库的连接后，可以向数据库发送 SQL 语句并且执行。要执行 SQL 语句首先要获得 `Statement` 类对象，通过获取 `Connection` 的 `prepareStatement` 方法来获取 `PreparedStatement` 对象。使用 `PreparedStatement` 来发送、执行 SQL 语句，例如：

```
// 获取数据库连接
Connection conn = MyConnection.getConnection();
// 获取 SQL 语句
String sql = MyConnection.getSQL();
PreparedStatement pstmt = null;
ResultSet rs = null;
try {
    pstmt = conn.prepareStatement(sql);
    rs = pstmt.executeQuery();
    //省略部分代码
} catch (Exception e) {
    e.printStackTrace();
}
```

### 3. 获取查询结果集

查询结果集是指将数据表中的数据查询出来保存在集合中，JDBC 中用于保存查询结果集的对象为 `ResultSet` 对象，例如，如下代码使用 JDBC 获取 `tb_Omnip` 表中所有的数据集合：

```
String strSql = "select * from tb_Omnip";
ResultSet res = null;
res = sql.executeQuery(str);
```







运行上述代码, 可将数据表 tb\_Omnip 中的数据全部检索出来, 保存在 ResultSet 对象中。通过遍历 ResultSet 对象, 获取数据表 tb\_Omnip 中的全部信息。

## 12.2.2 查询语句结构

调用程序向多功能查询模块中传输调用程序操作的数据库, 以及执行数据库操作的 SQL 语句。查询模块实现了向调用程序返回用户添加查询条件后的 SQL 语句。熟悉 SQL 查询语法至关重要, 本节将介绍 SELECT 查询语句的常用语法结构。

### 1. 简单的查询语句

简单的查询语句是指没有任何查询条件的 SQL 语句, 具体语法为:

```
SELECT 字段名 FROM 表名
```

“字段名”用于表示要进行查询的字段名称; “表名”用于指定要进行查询的数据表。例如, 本模块中查询数据表 tb\_Omnip 中的全部数据信息, SQL 语句如下:

```
String strSql = "select * from tb_Omnip";
```



上例中的 “\*” 号代表查询数据表 tb\_Omnip 中所有字段信息。

### 2. 带查询条件的 SQL 查询语句

通过使用带查询条件的 SQL 语句, 可以获取满足某一条件的记录, 条件查询是在 SELECT、FROM 子句后添加 WHERE 子句来进行条件查询, 具体语法为:

```
SELECT 字段名 FROM 表名 WHERE 查询条件
```

例如, 查询数据表 tb\_Omnip 中编号为 2 的记录, SQL 语句如下:

```
select * from tb_Omnip where id = 2;
```

### 3. 模糊查询

使用模糊查询可以搜索一些不确定条件的记录。模糊查询使用的操作符为 like, like 运算符可以与通配符一起使用。例如, 使用通配符%, 可以表示任意数量的字符; 下画线则表示一个字符。如果查询 tb\_Omnip 数据表中姓名格式以“张”开头的记录, SQL 语句如下:

```
select * from tb_omnip where name like '张%'
```

### 4. 多条件查询

通过多个条件查询可以获取多个组合条件的记录, 可以把逻辑运算符 AND、OR 结合搜索条件一块使用。在 WHERE 子句中使用 AND 运算符时, AND 运算符结合的两个条件都必须计算为 TRUE, 否则结果中不会包含这个记录。在 WHERE 子句中使用 OR 运算符时, 只要两个条件有一个为 TRUE, 结果就会包含这条记录。

例如, 查询 tb\_Omnip 数据表中 id 大于 2, 并且 name 等于李四的记录信息, SQL 语句如下:

```
select * from tb_omnip where id > 2 and name = '李四'
```





例如，查询 tb\_Omnip 数据表中 id 大于 2，或者 name 等于李四的记录信息，SQL 语句如下：

```
select * from tb_omnip where id > 2 or name = '李四'
```

### 12.2.3 获取字段的描述信息

获取 SQL Server 2005 数据库中数据表字段的描述信息，主要用到了数据库的系统表 sysobjects、sys.extended\_properties 和 syscolumns。

系统表 sysobjects 用于记录在数据库内创建的每个对象（约束、默认值、日志、规则、存储过程等），该表中的 name 字段记录了所有对象的名称。

系统表 sys.extended\_properties 用于保存关于运行在 Microsoft SQL Server 上的进程信息，这些进程可以是客户端进程或系统进程。该表中的 minor\_id 字段记录了所有表的字段 ID 号，value 字段记录所有表字段的描述信息。

系统表 syscolumns 用于记录每个表和视图中的每列，以及存储过程中的每个参数，该表位于每个数据库中。该表中的 name 字段记录了所有表的记录名。

获取 db\_Omnipeat 数据库中 tb\_Omnip 表的字段描述信息，代码如下：

```
use db_Omnipeat
select c.name,b.value FROM sysobjects a,sys.extended_properties b,syscolumns c
where a.id=b.major_id and b.minor_id=c.colorder
```

运行上段代码和获取数据表 tb\_Omnip 中字段的描述信息，如图 12.4 所示。

	name	value
1	id	编号
2	name	姓名
3	sex	性别
4	age	年龄
5	note	备注

图 12.4 tb\_Omnip 数据表字段的描述信息

### 12.2.4 获取数据库中的所有表名

在调用多功能查询模块的程序中，需要将本程序操作的数据库名称传递给多功能查询模块，多功能查询模块将根据数据库名称查询数据库中数据表的相应信息。首先应该完成查询数据库中的所有数据表，再根据获取的数据表集合查询相应的数据信息。java.sql 包中的 DatabaseMetaData 接口提供了获取数据库综合信息的方法。

DatabaseMetaData 接口由驱动程序供应商实现，在此让用户了解 Database Management System (DBMS)在与驱动程序相结合时的能力，不同的关系数据库管理系统常常支持不同的功能，以不同方式实现这些功能，并使用不同的数据类型。此外，驱动程序可以实现







DBMS 提供的顶级功能。

使用该接口中的 `getTables()` 方法, 可获取指定数据库中的所有数据表名, `getTables()` 方法返回 `ResultSet` 数据结果集, 具体语法如下:

```
getTables(String catalog, String schemaPattern, String tableNamePattern,
String[] types)
```

该方法的参数说明如表 12.1 所示。

表 12.1 `getTables()` 方法参数说明

参数	类型	描述
catalog	String	类别名称, 必须与存储在数据库中的类别名称匹配
schemaPattern	String	模式名称, 必须与存储在数据库中的模式名称匹配
tableNamePattern	String	表名称模式, 必须与存储在数据库中的表名称匹配
types	String[]	要包括的表类型所组成的列表

在多功能查询模块中, 通过 `getTables()` 方法获取指定数据表中的所有表名称, 关键代码如下:

```
try {
    String[] tableType = {"TABLE"}; //指定要进行查询的表类型
    Connection conn = MyConnection.getConnection(); //调用获取数据库连接的方法
    //获取 DatabaseMetaData 实例
    DatabaseMetaData databaseMetaData = conn.getMetaData();
    ResultSet resultSet = databaseMetaData.getTables(null, null, "%", tableType);
    //获取数据库中所有数据表集合
    //遍历集合
    while(resultSet.next()){
        String tableName = resultSet.getString("TABLE_NAME"); //见说明
    }
} catch (Exception e) {
    e.printStackTrace();
}
```



`getTables()` 方法获取的 `ResultSet` 数据库集合表由多个列组成, 其中, 列名称为 "TABLE\_NAME" 的数据列用来存储数据库中的所有数据表集合。

### 12.2.5 如何将程序加载到其他程序中

完成多功能查询模块, 可以将程序打包成 jar 文件, 方便随时调用。调用多功能查询模块的主程序需要通过 `BufferedWriter` 类, 将本程序操作的数据库、查询数据库所用的 SQL 语句写入 txt 文件中 (写入文件的顺序不可变), 并通过重写 `QueryFrame` 类中的 `clearButtonActionPerformed()` 方法, 可修改多功能查询模块窗体关闭后启动的窗体对象。

本节以 Eclipse 开发工具为例, 介绍如何将多功能查询模块 jar 文件添加到任意项目中, 具体步骤如下。

(1) 在主项目中, 单击鼠标右键, 在弹出的快捷菜单中选择“构建路径”\“配置构建路径”, 命令如图 12.5 所示, 选择要进行配置的文件路径。



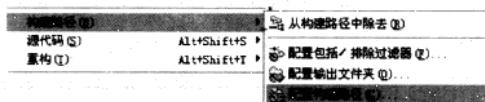


图 12.5 配置构建路径

(2) 单击“配置构建路径”按钮，将弹出如图 12.6 所示的项目属性对话框。

(3) 在项目属性对话框中单击“添加库”按钮，在弹出的“添加库”对话框中选择“用户库”，单击“下一步”按钮，如图 12.7 所示。

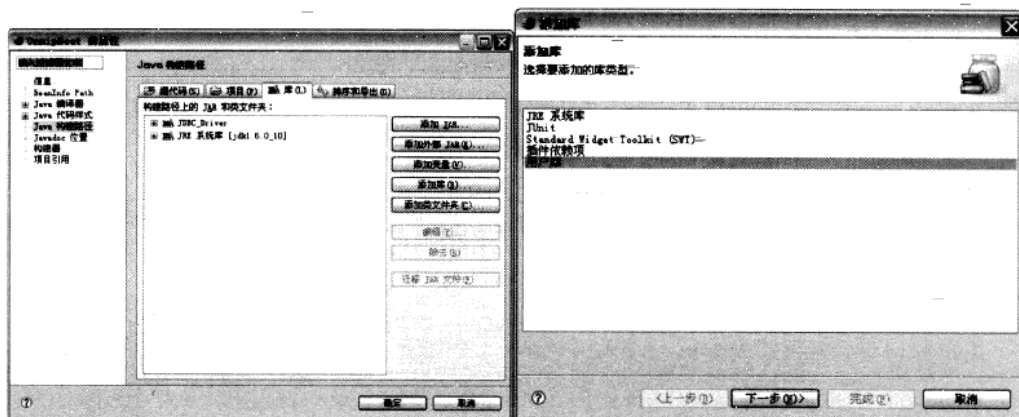


图 12.6 项目属性对话框

图 12.7 “添加库”对话框

(4) 在“添加库”对话框中，单击“用户库”，在弹出对话框中单击“新建”按钮，新建用户库，如图 12.8 所示。

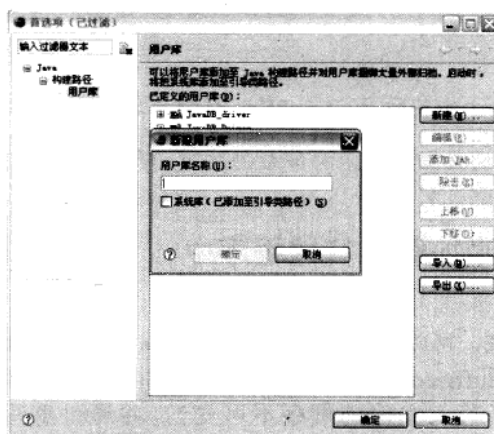


图 12.8 新建用户库

(5) 在“新建”用户库对话框中输入用户库名称后，可选择新建的用户库，单击图 12.8 中的“添加 JAR”按钮。在弹出的“文件选择”对话框中，选择多功能查询模块的





jar 文件所在的路径, 将其添加到用户库中, 单击“确定”按钮。完成将 jar 文件添加到项目中。

## 12.3 文件操作

### 12.3.1 功能概述

多功能查询模块的原理是使用界面的各项操作对字符串进行拼接, 最后形成一个可以执行的 SQL 语句。在拼接 SQL 的时候需要把字符串从文件中读取和写入, 这就需要程序对文件进行操作。操作文件的时候使用 File、FileWriter、FileReader、BufferedWriter 和 BufferedReader 类。

### 12.3.2 实现向 txt 文件中写数据

要调用多功能查询模块, 首先要将本程序所操作的数据库名称、SQL 语句写入到 txt 文件中。多功能查询模块将根据 txt 文件中的信息建立数据库连接, 并将用户添加的查询条件传递给调用程序。调用程序使用 BufferedWriter 类实现向 txt 文件中分行写入数据。本节将介绍在调用程序中如何向 txt 文件写数据, 以及如何接收多功能查询模块向调用程序传递的信息。

例如, 调用程序要查询数据库 db\_Omnipeat, 并将数据表 tb\_Omnip 中的全部信息显示在表格中, 因此需要向 txt 文件中写入 db\_Omnipeat, 以及查询数据表所用的 SQL 语句, 具体代码如下:

```
String strSql = "select * from tb_Omnip";
File file = new File("saveSQL.txt");           //创建 File 对象
String content[] = { "db_Omnipeat", strSql };   //定义字符串数组
if(!file.exists()){                             //如果该文件不存在
    file.createNewFile();                       //创建文件
    try {
        FileWriter fw = new FileWriter(file);   //创建 FileWriter 类对象
        BufferedWriter bufw = new BufferedWriter(fw); //创建 BufferedWriter 类对象
        for (int k = 0; k < content.length; k++) { //循环遍历数组
            bufw.write(content[k]);              //将字符串数组中的元素写入到磁盘文件中
            bufw.newLine();                      //将数组中的单个元素以单行的形式写入文件
        }
        bufw.close();                          //将 BufferedWriter 流关闭
        fw.close();                            //将 FileWriter 流关闭
    } catch (Exception e) {                    //处理异常
        e.printStackTrace();
    }
    .....
}
```

//省略部分代码

上述代码实现了在 saveSQL.txt 文件不存在的情况下, 新建文件并向文件中写数据, 写完数据之后, 多功能查询模块会将用户在模块窗体操作的查询条件写入 saveSQL.txt 文





件。为保证用户下一次调用多功能查询模块后，多功能查询模块会显示信息的准确性，需要将 saveSQL.txt 文件删除，并重新创建。具体代码如下：

```
String strSql = "select * from tb_Omnip";
File file = new File("saveSQL.txt");
String content[] = { "db_Omnipeat", strSql }; //定义字符串数组
if (file.exists()) {
    try {
        FileReader fr = new FileReader(file); //创建 FileReader 类对象
        BufferedReader bufr = new BufferedReader(fr); //创建 BufferedReader 类对象
        String s = null;
        int i = 0; //声明 int 型变量
        while ((s = bufr.readLine()) != null) { //如果文件的文本行数不为 null, 则进入循环
            str = s;
        }
        bufr.close(); //将 BufferedReader 流关闭
        fr.close(); //将 FileReader 流关闭
    } catch (Exception e) { //处理异常
        e.printStackTrace();
    }
    file.delete(); //将文件删除
    file.createNewFile(); //新建文件
    try {
        FileWriter fw = new FileWriter(file); //创建 FileWriter 类对象
        BufferedWriter bufw = new BufferedWriter(fw); //创建 BufferedWriter 类对象
        for (int k = 0; k < content.length; k++) { //循环遍历数组
            bufw.write(content[k]); //将字符串数组中的元素写入到磁盘文件中
            bufw.newLine(); //将数组中的单个元素以单行的形式写入文件
        }
        bufw.close(); //将 BufferedWriter 流关闭
        fw.close(); //将 FileWriter 流关闭
    } catch (Exception e) { //处理异常
        e.printStackTrace();
    }
}
```

在上段代码中，调用程序接收多功能查询模块传递的 SQL 语句，并将查询结果显示在界面中。需要注意的是，要将多功能查询模块返回的 SQL 语句删除，需要重新向 saveSQL.txt 文件写入数据，这样可以保证每次调用多功能查询模块时，系统不会在上次的查询条件基础上建立查询条件。

### 12.3.3 实现将查询结果写入 txt 文件中

如果用户指定一个完整的查询条件，必须要在“查询值”文本框中添加数值。可以在“查询值”文本框的焦点事件中将用户选择的查询条件写入 txt 文件中，然后可以在调用程序中检索 txt 文件中的信息。在“查询值”文本框的失去焦点事件中，首先判断用户是否选择合法的查询条件，具体代码如下：

```
/**
 * @param evt 失去焦点事件
 */
private void jTextField1FocusLost(java.awt.event.FocusEvent evt) {
    //判断用户是否选择正确的查询条件的字段
    if (!conditionComboBox.getSelectedItem().toString().equals("")
        && (!jComboBox4.getSelectedItem().toString().equals(""))) {
```





```

int n = JOptionPane.showConfirmDialog(this, "确定输入的字段名和关系正确吗?",
    "消息对话框", JOptionPane.YES_NO_OPTION);
if (n == JOptionPane.NO_OPTION) {
    conditionComboBox.setSelectedIndex(0);
    jComboBox4.setSelectedIndex(0);
    findjButton.setEnabled(false);           // “查看”按钮不可选
    valueTextField.setText("");              // 将查询值文本框清空
    return;
}
if (n == JOptionPane.YES_OPTION) {
    findjButton.setEnabled(true);
    strBuff.append(conditionComboBox.getSelectedItem().toString()
        + jComboBox4.getSelectedItem().toString()
        + valueTextField.getText());
}
}
//如果用户没有指定查询的字段, 将给出提示信息
if (conditionComboBox.getSelectedIndex() == 0) {
    JOptionPane.showMessageDialog(this, "没有确定查询字段", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
    findjButton.setEnabled(false);
    valueTextField.setText("");
    return;
}
//如果用户没有指定查询的关系, 或者用户没有选择“模糊查询”执行代码
if ((jComboBox4.getSelectedIndex() == 0)
    && !(jComboBox1.getSelectedItem().toString().equals("模糊查询"))) {
    JOptionPane.showMessageDialog(this, "没有确定查询关系", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
    findjButton.setEnabled(false);
    valueTextField.setText("");
    return;
}
File file = new File("saveSQL.txt");
String sql;
String str = null;
try {
    try {
        FileReader fr = new FileReader(file);           // 创建 FileReader 类对象
        // 创建 BufferedReader 类对象
        BufferedReader bufr = new BufferedReader(fr);
        int i = 0; // 声明 int 型变量
        // 如果文件行数不为 null, 则进入循环
        while ((sql = bufr.readLine()) != null) {
            i++;                                         // 将变量做自增运算
            str = sql;
        }
        bufr.close();                                  // 将 BufferedReader 流关闭
        fr.close();                                    // 将 FileReader 流关闭
    } catch (Exception e) {                            // 处理异常
        e.printStackTrace();
    }
    MyConn myConn = new MyConn();                      // 创建数据库连接类对象
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    Connection conn = myConn.getConnection();
    DatabaseMetaData meta = conn.getMetaData();
    // 用户选择的查询条件的“字段名”
    String name = conditionComboBox.getSelectedItem().toString();
    java.util.List list = getdate();
    String tableName = "";
    if (!list.isEmpty() && (list.size() > 0)) {
        for (int i = 0; i < list.size(); i++) {
            tableName = list.get(i).toString();
            // 根据字段的描述信息, 查询字段值
            String strSQL = "select c.name,b.value FROM sysobjects a,sysproperties
                b,syscolumns c "
                + "where a.name='"
                + tableName
                + "' and a.id=b.id and b.id=c.id and b.smallid=c.colorder
                and b.value ='"

```





```
        + name + "'";
        pstmt = conn.prepareStatement(strSQL);
    }
    rs = pstmt.executeQuery();
    while (rs.next()) {
        ResultSet rsr = meta.getColumns(null, null, tableName, rs
            .getString("name"));
        while (rsr.next()) {
            String strName = rsr.getString(6);
            if (strName.substring(0, 3).equals("int")) {
                try {
                    Integer inte = Integer.parseInt(valueTextField
                        .getText());
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(this, "请输入数值型数据",
                        "消息对话框", JOptionPane.WARNING_MESSAGE);
                    valueTextField.setText("");
                }
            }
        }
    }
    //如果用户指定“逻辑关系”，追加查询条件
    if ((str.indexOf("where") != -1)
        && (logicComboBox.getSelectedItem().toString()
            .equals("并且"))) {
        str = str + " and " + rs.getString("name")
            + jComboBox4.getSelectedItem().toString() + "'";
        valueTextField.setText("");
        System.out.println("SQLA|SQL =" + str);
    } else if ((str.indexOf("where") != -1)
        && (logicComboBox.getSelectedItem().toString()
            .equals("或者"))) {
        str = str + " or " + rs.getString("name")
            + jComboBox4.getSelectedItem().toString() + "'";
        valueTextField.setText("");
    } else if ((str.indexOf("where") == -1)
        && (jComboBox1.getSelectedItem().toString()
            .equals("模糊查询"))) {
        str = str + " where " + rs.getString("name") + " like "
            + "'" + valueTextField.getText() + "'";
    } else if ((str.indexOf("where") == -1)
        && (logicComboBox.getSelectedItem().toString()
            .equals(""))) {
        str = str + " where " + rs.getString("name")
            + jComboBox4.getSelectedItem().toString() + "'";
        valueTextField.setText("");
    }
    //将用户添加的查询条件写入文本文件中
    try {
        FileReader fr = new FileReader(file); //创建 FileReader 类对象
        // 创建 BufferedReader 类对象
        BufferedReader bufr = new BufferedReader(fr);
        int i = 0;
        String s = null;
        while ((sql = bufr.readLine()) != null) {
            //如果文件的文本行数不为 null，则进入循环
            s = sql;
            break;
        }
        String content[] = new String[2]; //定义字符串数组
        content[0] = s;
        content[1] = str;
        //创建 FileWriter 类对象
        FileWriter fw = new FileWriter(file);
        //创建 BufferedWriter 类对象
        BufferedWriter bufw = new BufferedWriter(fw);
        for (int k = 0; k < content.length; k++) { //循环遍历数组
            //将字符串数组中的元素写入到磁盘文件中
            bufw.write(content[k]);
            bufw.newLine(); //将数组中的单个元素以单行的形式写入文件
        }
        bufw.close(); //将 BufferedWriter 流关闭
    }
```







```

        fw.close(); //将 FileWriter 流关闭
    } catch (Exception e) {
        e.printStackTrace();
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

```

## 12.4 事件处理

### 12.4.1 功能概述

多功能查询模块通过界面的操作拼接 SQL 语句,当使用者在界面上做选择、提交等操作的时候程序自己会触发一系列的事件、执行某些方法。对使用者触发的方法做相对应的处理,如当使用者单击“查看”按钮时,在“查看”按钮的单击事件中弹出选择条件的窗口;当使用者单击“关闭”按钮时,在“关闭”按钮的单击事件中做关闭当前窗口的操作。

### 12.4.2 实现获取表中的字段描述信息

大多数据库表采用英文字段,如果将英文字段作为多功能查询模块中的查询条件,不方便用户使用。基于这种情况,多功能查询模块的设计把表中的字段描述信息用来建立查询条件,将查询得出的字段描述信息添加到下拉列表中,如图 12.9 所示。

首先应该获取调用程序操作的数据表中的字段描述信息,并将描述信息添加到下拉列表中,具体代码如下:

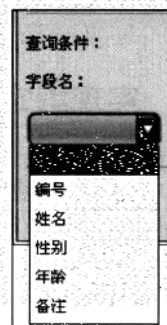


图 12.9 获取表中的字段描述信息

```

File file = new File("saveSQL.txt");
String s = null;
String stri = null;
try {
    FileReader fr = new FileReader(file); //创建 FileReader 类对象
    BufferedReader bufr = new BufferedReader(fr); //创建 BufferedReader 类对象
    int i = 0; //声明 int 型变量
    while ((s = bufr.readLine()) != null) { //如果文件的文本行数不为 null,则进入循环
        i++; //将变量做自增运算
        stri = s;
    }
    bufr.close(); //将 BufferedReader 流关闭
    fr.close(); //将 FileReader 流关闭
} catch (Exception e) {
    e.printStackTrace(); //处理异常
}
try {
    pstmt = conn.prepareStatement(stri);
    rs = pstmt.executeQuery();
}

```







```
int count = 0;
if (rs.next()) {
    ResultSetMetaData rsmd = rs.getMetaData();
    count = rsmd.getColumnCount();
    DatabaseMetaData databaseMetaData = conn.getMetaData();
    String[] tableType = { "TABLE" };
    ResultSet resultSet = databaseMetaData.getTables(null, null,
        "%", tableType); //获取数据库中的所有表集合
    try {
        while (resultSet.next()) {
            String tableName = resultSet.getString("TABLE_NAME");
            String strSQL = "select c.name,b.value FROM sysobjects
                a,sysproperties b,syscolumns c "
                + " where a.name='"
                + tableName
                + "' and a.id=b.id and b.id=c.id and
                b.smallid=c.colorder";
            pstmt = conn.prepareStatement(strSQL);
            rs = pstmt.executeQuery();
            int i = 1;
            String[] str = new String[count + 1];
            str[0] = "";
            while (rs.next()) {
                str[i] = rs.getString("value"); //循环为 String 数组赋值
                i = i + 1;
            }
            conditionComboBox
                .setModel(new javax.swing.DefaultComboBoxModel(
                    str)); //在下拉列表中添加数据
        }
    } finally {
        resultSet.close();
    }
}
```

### 12.4.3 实现绑定组件的处理事件

在多功能查询模块中后“查询分类”下拉列表中允许用户选择“顺序查询”、“模糊查询”。当用户选择“模糊查询”后，根据模糊查询的语法规则，用户不需要指定查询“关系”，因此“关系”下拉列表不允许用户编辑，如图 12.10 所示。

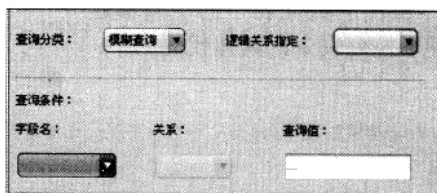


图 12.10 查询分类

“查询分类”下拉列表的单击事件具体代码如下：

```
jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "", "
    顺序查询", "模糊查询" }));
jComboBox1.addActionListener(new java.awt.event.ActionListener() {
    //绑定单击事件
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jComboBox1ActionPerformed(evt);
    }
});
```





```
private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt){
    if( jComboBox1.getSelectedItem().toString().equals("模糊查询")){
        jComboBox4.setEnabled(false);           //“关系”下拉列表不可选
        JOptionPane.showMessageDialog(this, "选择模糊查询, 可在'查询值'一栏中输入'_'
        或'%'等。如'张_'、'张%'", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
    }
    else{
        //如果用户没有选择“模糊查询”,“关系”下拉列表可选
        jComboBox4.setEnabled(true);
    }
}
```

多功能查询模块允许用户进行多条件查询。例如, 查询编号大于3, 并且姓名等于“张三”的员工信息, 可以通过“逻辑关系”下拉列表指定查询条件。“逻辑关系”下拉列表中包含“并且”、“或者”两项内容。在用户选择“逻辑关系”前必须要有一个查询条件, 也就是说用户必须要在“查询值”文本框中添加查询条件。“逻辑关系”下拉列表的单击事件代码如下:

```
//变量 logicComboBox 为“逻辑关系”下拉列表
logicComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[]
{ "", "并且", "或者"}));
logicComboBox.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jComboBox2ActionPerformed(evt); //调用 jComboBox2ActionPerformed 方法
    }
});
```

当用户确定“查询值”后, 系统会将用户添加的查询信息写入文本文件中, 以便向调用程序传递参数, 因此需要向“查询值”文本框添加焦点事件, 具体代码如下:

```
// valueTextField 为“查询值”文本框
valueTextField.addFocusListener(new java.awt.event.FocusAdapter(){
    public void focusGained(java.awt.event.FocusEvent evt) {
        jTextField1FocusGained(evt);
    }
    public void focusLost(java.awt.event.FocusEvent evt) {
        jTextField1FocusLost(evt);
    }
});
```

#### 12.4.4 显示调用程序窗体

当用户单击查询窗体中的“提交”按钮后, 系统将显示返回调用窗体的界面。用户可以通过重写“提交”按钮的单击事件来处理一些后续工作, 释放窗体资源, 如重新显示调用窗体的界面。“提交”按钮的单击事件代码如下:

```
private void tijiaoButtonActionPerformed(java.awt.event.ActionEvent evt) {
    QueryFrame queryFrame = new QueryFrame(); //创建调用程序对象
    queryFrame.setVisible(true);
    this.dispose(); //释放本窗体资源
}
```



# 第 13 章

---

## 文件分割模块

(Swing+I/O 文件处理技术实现)

随着计算机逐步深入人们的生活，人们开始将越来越多的文件存放到计算机上，这就要求有一款功能强大的文件处理软件能帮助人们处理计算机中的文件。为了适应现代社会发展的需要，更大程度地满足广大用户的文件处理需求，本章使用 Java 语言制作了一个功能强大的文件分割模块，用户可以通过该模块执行对文件、文件夹的批量复制、剪切、删除、重命名、搜索、解压缩、分割合成等操作。通过本章的学习，读者能够学到：

- » 遍历系统文件夹
- » 复制文件的方法
- » 文件分割与合并的方法
- » 对文件编码格式的转换
- » 实现文件分类管理
- » 实现文件的压缩与解压缩





## 13.1 文件分割模块概述

### 13.1.1 模块概述

由于计算机的普及,人们将越来越多的文件存储在计算机上,如果没有定期地对系统文件进行分类,删除过期文件,系统文件会越来越多,不方便查找。文件分割模块是为了方便用户管理文件而设计的,通过本模块可快速地实现文件的批量复制、批量删除、批量重命名、文件分类等。此外,为了方便用户查看系统文件,提供了文件搜索的功能。对于较大的文件,对其进行存储、移动都不是很方便,因此设计了文件的分割、合并子模块,通过该子模块,可以对大文件进行分割,同时也可以对分割的文件进行合并,这样不会影响文件的性能。

### 13.1.2 功能结构

文件分割模块可以实现新建文件、对文件进行分类和搜索、批量复制、移动、删除和重命名文件,以及对文件进行分割与合并、压缩和解压缩等操作,其功能结构如图 13.1 所示。

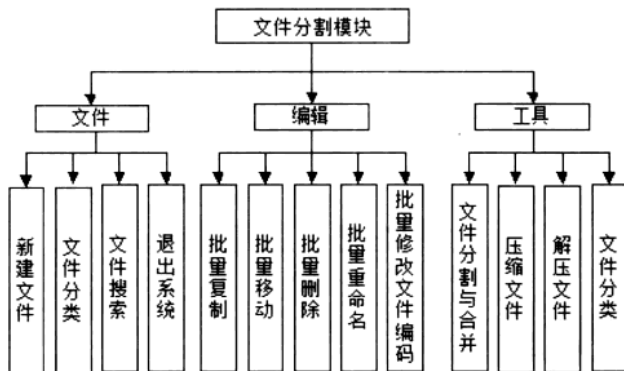


图 13.1 模块功能结构图

### 13.1.3 程序预览

为了使读者更了解本模块,下面给出本模块的几个典型界面。在主窗体中提供了进入其他功能模块的窗口,并提供了类似包资源管理器的功能结构,通过该结构,可实现对系统文件的复制、移动、删除等操作。主窗体如图 13.2 所示。

通过单击“编辑”菜单中的“批量修改文件编码”菜单项,可实现对文件编码格式的





批量修改，界面如图 13.3 所示。

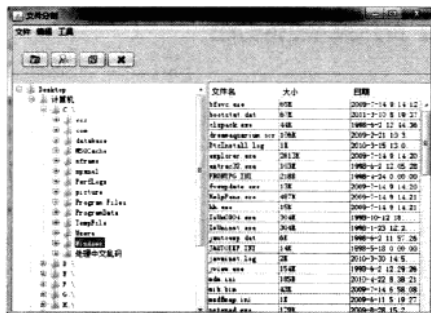


图 13.2 主窗体图

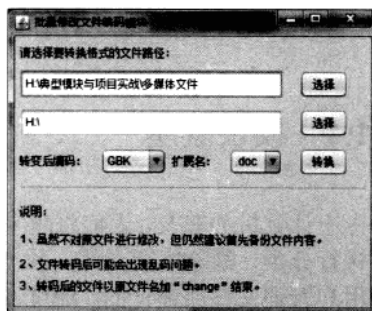


图 13.3 批量修改文件编码界面

用户通过单击“工具”菜单中的“分割/合并”菜单项，可实现对指定文件的分割与合并，界面如图 13.4 所示。

通过文件搜索模块可以实现查找系统指定盘符中指定后缀名的文件，并可以显示文件总数，界面如图 13.5 所示。

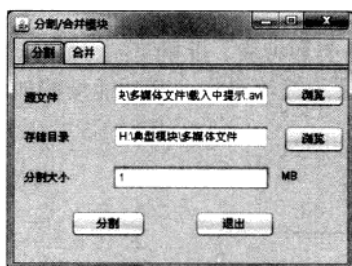


图 13.4 文件分割/合并界面

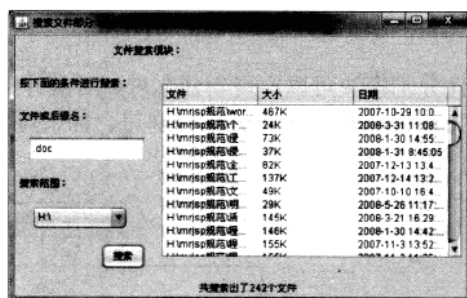


图 13.5 搜索文件界面

## 13.2 关键技术

### 13.2.1 文件操作与读写

文件分割模块在对文件及文件夹进行批量处理时，主要用到了输入/输出流技术，操作文件流的类都位于 `java.io` 包中。本模块中主要用到了 `File` 类、`FileInputStream` 类、`FileOutputStream`。下面对这 3 个类及重要方法进行详细介绍。

#### 1. File 类

`File` 类定义了一些与平台无关的方法来操作文件，可以通过调用 `File` 类中的方法实现创建、删除、重命名文件等。

`File` 类提供了多种构造函数，本模块中主要应用了如下构造函数来创建 `File` 对象。

语法：





```
File file = new File(String pathName);
```

说明: pathName 用来指定路径名称, 例如:

```
File file = new File("H://12.txt");
```

File 类中提供了文件操作的常用方法, 例如, 本模块中用到了文件删除、文件重命名、新建文件等方法。

#### 1) createNewFile()方法

语法:

```
createNewFile()
```

说明: 如果指定抽象路径中不包含指定文件, 该方法会创建文件。如果抽象路径中包含指定文件, 则不创建文件。

例如, 本模块中实现新建文件方法时首先判断指定文件目录下是否有指定文件存在, 如果没有则调用 createNewFile()方法创建文件, 关键代码如下:

```
File file = new File(myFileName); //myFileName 为 String 对象, 表示创建 File 对象路径
if (!file.exists()) {
    file.createNewFile();
}
mkdir()方法
语法:
mkdir()
```

说明: 该方法用于创建此抽象路径中指定的文件目录, 通常创建文件夹时使用该方法, 本模块中创建文件夹通过该方法实现, 关键代码如下:

```
String filePath = strPath;
File myFilePath = new File(filePath); //根据参数 filePath 创建 File
if (!myFilePath.exists()) {
    myFilePath.mkdir();
}
```

#### 2) delete()方法

语法:

```
delete()
```

说明: 该方法用于删除文件或文件夹, 如果要删除指定文件夹, 必须保证删除的文件夹为空, 才能进行删除。例如, 本模块中删除整个文件夹的方法为 deleteDirs(), 关键代码如下:

```
/**
 * @param file 要删除的文件对象
 */
public void deleteDirs(File file) {
    if (file.exists()) {
        if (file.isFile()) {
            file.delete(); //如果该文件存在
                           //并且是文件对象
                           //将文件删除
        }
        else if (file.isDirectory()) {
            File[] files = file.listFiles(); //如果该文件对象是路径
            for (int i = 0; i < files.length; i++) { //遍历文件中的文件对象
                this.deleteDirs(files[i]); //重新调用本方法
            }
            file.delete();
        }
    }
}
```





### 3) listFiles()方法

语法:

```
listFiles()
```

说明: 该方法返回一个抽象路径名的数组, 这些路径名表示此抽象路径名的文件名称。例如, 在本模块中检索某文件夹下的所有文件方法时, 利用 listFiles()方法获取指定文件夹下的所有文件, 关键代码如下:

```
LinkedList filelist = new LinkedList();           //创建 List 实例
File dir = new File(strPath);                     //根据 strPath 对象创建 File 对象
File[] file = dir.listFiles();                     //获取 File 路径中的文件集合
if(file.length > 0){
    for (int i = 0; i < file.length; i++) {        //循环遍历数组
        if (file[i].isDirectory()) {               //如果数组中的某个元素是一个路径
            getFileList(file[i].getAbsolutePath()); //重新调用该方法
        } else {                                   //否则, 将 File 数组中的元素添加到集合中
            filelist.add(file[i]);
        }
    }
}
```

### 4) renameTo() 方法

语法:

```
renameTo(File dest)
```

说明: 该方法用于重新命名此抽象路径名表示的文件, 参数 dest 用于指定文件的新抽象路径名。本模块中实现文件重命名通过调用 renameTo()方法实现, 关键代码如下:

```
public void renamePath(String oldpath, String newPath) {
    File file = new File(oldpath);                 //创建重命名前的文件对象
    File files = new File(newPath);                 //创建重命名后的文件对象
    if (!files.exists()) {                           //如果指定重命名后的文件名在抽象路径中不存在
        file.renameTo(files);                       //可以实现重命名操作
    }
}
```

## 2. FileInputStream 类与 FileOutputStream 类

FileInputStream 类与 FileOutputStream 类提供了文件操作的基本功能, FileInputStream 类提供了读取文件的能力, FileOutputStream 类提供了写入文件的基本能力。FileInputStream 类通过 read()方法读取文件信息, FileOutputStream 类通过 write()方法向文件写入内容。

read()方法有 3 种重载形式。

### 1) read()方法

语法:

```
read()
```

说明: 从输入流中读取一个数据字节, 如果没有输入可用, 则此方法将阻塞。

### 2) read(byte[] b) 方法

语法:

```
read(byte[] b)
```

说明: 从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中, 在某些输入







可用之前，此方法将阻塞。

3) read(byte[] b, int off, int len) 方法

语法：

```
read(byte[] b, int off, int len)
```

说明：从此输入流中将最多 len 个字节的数据读入一个 byte 数组中。如果 len 不为 0，则在输入可用之前，该方法将阻塞；否则，不读取任何字节并返回 0。

例如，在本模块中实现复制文件时，首先通过 FileInputStream 类对文件进行读取，再通过 FileOutputStream 类将文件写入另一个文件中，关键代码如下：

```
//给定参数为要进行复制的文件，以及复制后的文件
public void copyFile(String oldPath, String newPath) {
    try {
        int bytesum = 0;
        int byteread = 0;
        File oldfile = new File(oldPath);           //创建 File 对象
        if (oldfile.exists()) {                     //文件存在时
            InputStream inStream = new FileInputStream(oldPath); //读入原文件
            FileOutputStream fs = new FileOutputStream(newPath); //写入文件
            byte[] buffer = new byte[1444];          //指定 byte 数组
            环获取要读入文件的字节大小
            while ((byteread = inStream.read(buffer)) != -1) {
                bytesum += byteread;
                fs.write(buffer, 0, byteread);        //写入文件
            }
            inStream.close();                        //关闭流
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 13.2.2 获取系统有效盘符

不同的计算机的系统盘符是不相同的，有的计算机有“C:、D:、E:、F:”盘。而有的计算机只有“C:、D:”盘，那么，如何获取本地有效盘符呢？本模块在文件搜索子模块中设计了按照本地指定盘符进行搜索文件，将检索出来的有效盘符添加到下拉列表中，

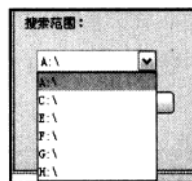


图 13.6 本地有效磁盘

如图 13.6 所示。

本实例通过 File 类的 listRoots()方法获取本地有效磁盘，关键代码如下：

```
public File[] getRoot(){
    File[] roots = File.listRoots();
    return roots;
}
```

### 13.2.3 转换文件编码格式

文件编码格式转换主要是为了方便跨平台操作而设计的。例如，有一些文件的编码格





式是“UTF-8”，这样的文件使用例如编码格式为“GBK”的文件编辑器，打开文件时会出现乱码情况。针对这种情况，文件分割模块设计了批量转换文件编码格式子模块来实现转变文件编码格式。本模块中的实现文件的编码格式的转换，并没有将原文件进行转换，是创建了一个新文件，用于保存编码转换后的新文件。

首先通过 `FileInputStream` 类将要进行转换的文件读取出来，再通过 `FileOutputStream` 类将文件以另一种编码格式写入到新文件中。下面是本模块中实现文件编码格式转换方法的代码，在该方法中包含 3 个参数，分别是要进行转换的文件地址、转换结束的文件保存地址、以及文件转换后的编码格式，具体代码如下：

```
public void setEnd(String fileStr, String fileSave, String change) {
    try {
        File file = new File(fileStr);           //创建要进行文件编码转换的 File 对象
        //读取 FileInputStream 对象
        FileInputStream fis = new FileInputStream(file);
        byte byt[] = new byte[1024];
        File filea = new File(fileSave);         //创建文件编码格式转换后的 File 对象
        filea.createNewFile();                   //新建文件
        //创建 FileOutputStream 对象
        FileOutputStream fop = new FileOutputStream(filea);
        String str = "";
        int read = -1;
        while ((read = fis.read(byt)) != -1) {    //获取要进行编码转换的文件字节数
            str = new String(byt, 0, read, change); //见说明
            fop.write(str.getBytes());             //写文件
        }
        fop.close();                             //关闭流
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

`String` 类提供了多种构造方法，可根据指定的字符集和指定的 `byte` 子数组，构造一个新的 `String` 对象。

语法：

```
String str = new String(byte[] bytes,int offset,int length,Charset charset);
```

参数如表 13.1 所示。

表 13.1 String 构造方法的参数说明

参数	类型	描述
bytes	byte[]	要解码为字符的 byte 型数组
offset	int	要解码的 byte 数组的第一个索引位置
length	int	要解码的 byte 数组的长度
charset	int	指定创建 String 的编码格式

例如，将编码格式是 UTF-8 的 `ZipFrame.java` 文件进行格式编码转换，可以看出转换前与转换后的区别，如图 13.7 与图 13.8 所示。



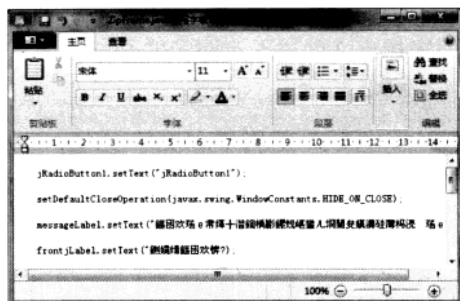


图 13.7 编码转换前的效果

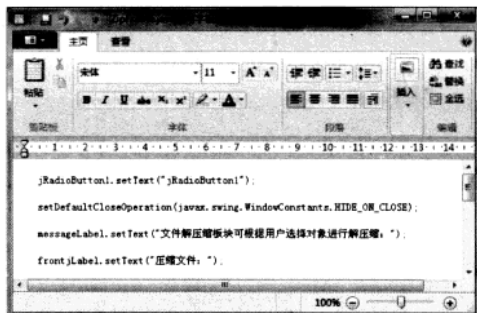


图 13.8 编码转换后的效果

### 13.2.4 文件解压缩

对文件进行解压缩可以节省磁盘空间, 因此, 在文件分割模块中添加了将文件压缩成 ZIP 文件格式的功能, 通过本模块还可以对 ZIP 文件的格式进行解压。本节将介绍 Java 中的 ZIP 解压缩技术。

在 Java 的内置类中, 提供了非常好用的相关类来实现 ZIP 解压缩。通过 java.util 包中的 ZipOutputStream 类可实现对文件的压缩, 通过 ZipInputStream 类可实现对文件解压处理。对于 ZIP 格式的文件, 尾端会有一个“目录进入点”, 以此来指定此 ZIP 文件内的各个区域容纳了哪个文件, 如图 13.9 所示。

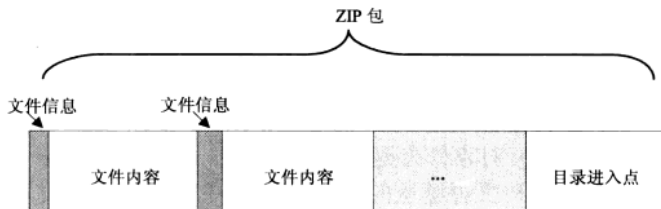


图 13.9 Zip 文件结构

如果要将某个文件写入到 ZIP 文件内, 必须先写入对应到该文件的“目录进入点”, 并且把要写入文件内容的位置移动到此进入点所指的位置, 再写入文件内容。若要从 ZIP 文件内读取某个文件, 要先找到对应文件的“目录进入点”, 然后从该进入点可知该文件在 ZIP 文件内的位置, 才能去读取这个文件内容。

#### 1. 压缩文件

ZipOutputStream 类可将文件压缩为 ZIP 文件, 该类构造函数语法如下:

```
ZipOutputStream(OutputStream out);
```

该类提供了一个重要的方法 putNextEntry(ZipEntry e), 用于写入新的 ZIP 文件条目并将流定位到条目数据的开始处, 语法如下:

```
putNextEntry(ZipEntry e)
```

说明: e 是 ZipEntry 对象, 用于表示 ZIP 文件条目。





在文件分割模块中，实现压缩文件的 zip() 方法中指定了两个参数，分别是 ZipOutputStream (要进行压缩的 File 对象)，以及用来指定文件进入点路径的 String 对象。关键代码如下：

```
public void zip(ZipOutputStream out, File f, String base) {
    try {
        if (f.isDirectory()) { //如果要进行压缩的文件是一个路径
            File[] fl = f.listFiles();
            out.putNextEntry(new ZipEntry(base + "/" + f.getName())); //获取文件的目录
            base = base.length() == 0 ? "" : base + "/"; //判断 base 对象是否为空
            for (int i = 0; i < fl.length; i++) {
                zip(out, fl[i], base + fl[i].getName());
            }
        } else {
            out.putNextEntry(new ZipEntry(base + f.getName()));
            FileInputStream in = new FileInputStream(f);
            strPath = f.getAbsolutePath();
            int b;
            while ((b = in.read()) != -1) {
                out.write(b); //向目录进入点写文件
            }
            in.close();
            out.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 2. 解压文件

ZipInputStream 类可实现文件的解压，该类的构造方法语法如下：

```
ZipInputStream(InputStream in)
```

该类提供了一个重要的方法 getNextEntry()，用于读取下一个 ZIP 文件条目，并将流定位到该条目数据的开始处，语法如下：

```
getNextEntry()
```

·说明：该方法以 ZipEntry 对象作为返回值。

文件分割模块的实现 ZIP 文件解压的 unzip() 方法中，有两个参数，分别是要进行解压的 ZIP 文件路径，解压后文件的保存路径，关键代码如下：

```
public void unzip(String zipFileName, String outputDirectory) {
    ZipInputStream in;
    try {
        in = new ZipInputStream(new FileInputStream(zipFileName));
        ZipEntry z;
        while ((z = in.getNextEntry()) != null) {
            System.out.println("unzipping " + z.getName());
            if (z.isDirectory()) //获取文件进入点是目录
            {
                String name = z.getName(); //获取路径名称
                name = name.substring(0, name.length() - 1);
                File f = new File(outputDirectory + File.separator + name);
                f.mkdir(); //创建抽象路径
            }
            else {
                File f = new File(outputDirectory + File.separator + z.getName());
                System.out.println("FILE " + outputDirectory + z.getName());
                f.createNewFile();
                FileOutputStream out = new FileOutputStream(f);
            }
        }
    }
}
```







```

        int b;
        while ((b=in.read()) != -1)
            out.write(b);
        out.close();
    }
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
}

```

### 13.2.5 表格控件的使用

本模块多处用到 `JTable` 组件来显示系统文件的属性, 使用 `JTable` 可以允许用户编辑数据, 但需要注意的是, `JTable` 并没有包含或缓存任何数据, 它只是数据的视图。如图 13.10 所示, 使用表格空间实现显示某文件夹下的文件名、文件大小、文件的创建日期等信息。

文件名	大小	日期
01.bmp	504K	2008-6-18 11:1...
02.bmp	315K	2008-6-13 13:3...
03.bmp	1228K	2008-6-15 8:45:31
04.bmp	1228K	2008-6-15 8:47:31
05.bmp	159K	2008-6-15 11:2...
06.bmp	400K	2008-6-15 15:4...
07.bmp	99K	2008-6-16 11:2...
08.bmp	548K	2008-6-16 13:1...
09.bmp	727K	2008-6-16 13:2...
10.bmp	99K	2008-6-16 14:1...
11.bmp	727K	2008-6-16 15:3...
12.bmp	321K	2008-6-16 16:1...

图 13.10 表格显示文件信息

`JTable` 提供了多个构造函数, 可方便地创建表格对象, 下面给出两种常见的构造函数。

1) `JTable(Object[][] rowData, Object[] columnNames)`

语法:

```
JTable(Object[][] rowData, Object[] columnNames)
```

说明: 构造一个 `JTable` 来显示二维数组 `rowData` 中的值, 其列名称为 `columnNames`, 所有行的长度必须与 `columnNames` 的长度相同。

2) `JTable(int numRows, int numColumns)`

语法:

```
JTable(int numRows, int numColumns)
```

说明: 使用 `DefaultTableModel` 构造具有 `numRows` 行和 `numColumns` 列的 `JTable`, 列名称采用 "A"、"B"、"C" 等形式。

虽然使用这些构造函数可以便于使用 `JTable` 组件, 但使用构造函数也具有有一些限制。例如, 通过第一种构造函数创建的表格组件, 表格中的数据类型会被看做是相同的 `String` 类型数据, 这样, 如果表格中具有 `boolean` 数据会将 `boolean` 数据显示成一个字符串, 或者使用构造函数创建的表格对象会自动使每一个单元格可编辑。如果想摆脱这些限制, 必须实现自己的表格模型。

使用表格模型可帮助表格以最合适的格式显示数据, 并且使用构造函数创建的表格对象, 各单元格是可编辑的, 而通过表格模型可以设置表格中的列是否可编辑。





在文件分割模块中创建表格模型关键代码如下：

```
Class[] types = new Class[] { java.lang.Object.class,
    /定义 Class 类型数组，保存列表中的数据
    java.lang.String.class, java.lang.String.class };
boolean[] canEdit = new boolean[] { false, false, false };
public LocalTableModel() {
    定义表格列名
    super(new Object[][] {}, new String[] { "文件名", "大小", "日期" });
}
public Class getColumnClass(int columnIndex) {
    return types[columnIndex];           //获取指定列的数据类型
}
public boolean isCellEditable(int rowIndex, int columnIndex) {
    return canEdit[columnIndex];         //设置单元格不可编辑
}
```

成功创建表格模型后，可根据定义的表格模型创建表格对象。在文件分割模块中实现创建 JTable 组件，并设置表格模型，关键代码如下所示：

```
private JTable getFileList() {
    if (fileList == null) {
        fileList = new JTable();          //实例化表格模型
    }
    fileList.setModel(new LocalTableModel()); //设置表格模型
    return fileList;
}
```

成功创建表格对象后，需要对表格进行维护，例如，向表格中添加新的数据行、修改表格中某一个单元格的值、清空表格等，这些操作可以通过维护表格模型来完成。

向表格中添加新的数据行，可使用 `addRow()` 方法完成。通过该方法的多种重载形式可实现将数据添加到表格尾部，或是表格中的指定位置。

1) `addRow(Object[] rowData)` 方法

语法：

```
addRow(Object[] rowData)
```

说明：将由数组封装的数据添加到表格模型的尾部。

2) `addRow(Vector rowData)` 方法

语法：

```
addRow(Vector rowData)
```

说明：将由向量封装的数据添加到表格模型的尾部。

3) `insertRow(int row, Object[] rowData)` 方法

语法：

```
insertRow(int row, Object[] rowData)
```

说明：将由数组封装的数据添加到表格模型的指定索引位置。

4) `insertRow(int row, Vector rowData)` 方法

语法：

```
insertRow(int row, Vector rowData)
```

说明：将由向量封装的数据添加到表格模型的指定索引位置。







## 13.3 主窗体

### 13.3.1 功能概述

文件分割模块的主窗体可以分为菜单栏部分、工具栏部分、树组件显示系统文件、表格组件显示系统文件信息，主窗体运行结果如图 13.11 所示。

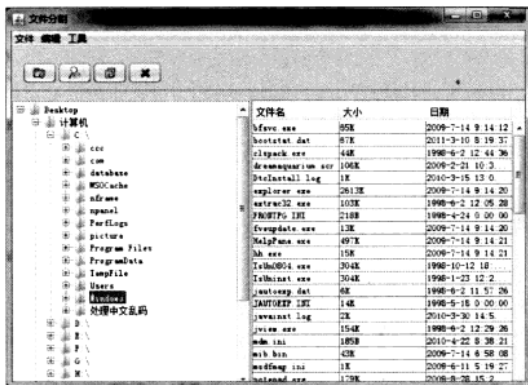


图 13.11 主窗体运行结果

### 13.3.2 菜单栏设计

在主窗体中共提供了 3 个菜单，分别为“文件”、“编辑”、“工具”。通过“文件”菜单可实现“新建文件”等功能，通过“编辑”菜单可实现文件的批量删除、复制、移动等功能，通过“工具”菜单可实现文件合并、分割、解压缩等功能。

下面以创建“文件”菜单为例，介绍实现菜单栏的步骤。

(1) 创建菜单栏、菜单、菜单项对象。

```
private JMenuBar jmenubar = null;           //创建菜单栏对象
private JMenu fileMenu = null;              //创建“文件”菜单对象
private JMenuItem createFile = null;        //创建“新建”菜单项对象
private JMenuItem findFileItem = null;      //创建“搜索”菜单项对象
private JMenuItem close = null;             //创建“退出”菜单项对象
```

(2) 实例化菜单对象，具体代码如下：

```
private void initialize() {
    jmenubar = new JMenuBar();
    fileMenu = new JMenu("文件");
    createFile = new JMenuItem("新建");
    findFileItem = new JMenuItem("搜索");
}
```

(3) 将菜单栏添加到窗体，将菜单添加到菜单栏中，并将菜单项添加到菜单中，具体代码如下：





```
private void initialize() {
    this.setJMenuBar(jmenubar);
    fileMenu.add(createFile);
    fileMenu.add(findFileItem);
    fileMenu.add(close);
}
```

(4) 为菜单项绑定单击事件。

```
private void initialize() {
    createFile.addActionListener(new FileMenu());
    findFileItem.addActionListener(new FileMenu());
    close.addActionListener(new FileMenu());
}
```



FileMenu 类为自定义处理菜单项的单击事件类，该类继承了 ActionListener 接口。

### 13.3.3 工具栏设计

工具栏为进入某个功能模块提供了快捷方法。在文件分割模块主界面中设计了工具栏，并在工具栏中添加了 4 个按钮，分别为“新建文件”按钮、“文件搜索”按钮、“文件分类”按钮、“退出”按钮。

下面以“新建文件”按钮为例，介绍在文件分割模块中实现工具栏的过程。

(1) 创建工具栏 JToolBar 对象，以及向工具栏中添加的按钮对象，具体代码如下：

```
private JToolBar jToolBarBar = null;
private JButton jButtonCreate = null;
```

(2) 创建 getJToolBar() 方法，该方法返回 JToolBar 对象，在该方法中实现实例化 JToolBar 对象，并实现向工具栏中添加按钮，具体代码如下：

```
private JToolBar getJToolBar() {
    if (jToolBarBar == null) {
        jToolBarBar = new JToolBar(); //实例化 JToolBarBar 对象
        jToolBarBar.setRollover(true); //绘制工具栏按钮边框
        //指定要设置按钮的图片
        URL url = getClass().getResource("/image/create.png");
        ImageIcon icon = new ImageIcon(url);
        jButtonCreate = new JButton(icon); //实例化“新建”按钮
        jButtonCreate.setFocusable(false); //设置焦点状态
        jButtonCreate
            .setHorizontalTextPosition(javax.swing.SwingConstants.CENTER); //设置文本相对于图标的水平位置
        jButtonCreate
            .setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM); //设置文本相对于图标的垂直位置
        jButtonCreate.setToolTipText("新建"); //注册要在工具提示中显示的文本
        jButtonCreate.addActionListener(new ToolButton()); //绑定按钮的单击事件
        jToolBarBar.add(jButtonCreate); //向工具栏中添加按钮
    }
    return jToolBarBar;
}
```







如上代码设置了工具栏的显示图标,当鼠标移动到工具栏上时,会显示工具栏中设置的显示文本,如图13.12所示。



图 13.12 工具栏

(3) 将工具栏添加到窗体中,完成这一功能,只需将 `getJToolBar()` 方法返回 `JToolBar` 对象,添加到主面板中即可,代码如下:

```
jContentPane.add(getJToolBar(), null);
```

### 13.3.4 实现显示系统文件夹

文件分割模块主窗体中使用 `JTree` 组件显示系统文件结构,类似于 Windows 系统中的包资源管理器。该 `JTree` 组件的根节点为“桌面”,如图13.13所示。通过本模块的 `JTree` 组件不仅可以检索本地系统中的文件,并且可以通过“网上邻居”浏览临近电脑。由于展开树组件需要很大空间,可以将其放在滚动窗格中。

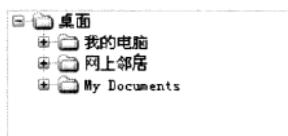


图 13.13 树结构中的根节点及各主要子节点

实现通过树组件来显示系统文件夹,首先应该创建 `MyNode` 类来设置树组件中的各个节点, `MyNode` 类继承自 `DefaultMutableTreeNode` 类。 `DefaultMutableTreeNode` 类是树数据结构中的通用节点,具体代码如下:

```
class MyNode extends DefaultMutableTreeNode {
    private boolean explored = false;
    public MyNode(File file) {
        setUserObject(file); //将此节点的用户对象设置为 file
    }
    public boolean getAllowChildren() { //如果允许此节点拥有子节点,则返回 true
        return isDirectory();
    }
    public boolean isLeaf() { //如果此节点没有子节点,则返回 true
        return !isDirectory();
    }
    public File getFile() { //获取该节点的用户对象
        return (File) getUserObject();
    }
    public boolean isExplored() {
        return explored;
    }
    public boolean isDirectory() { //判断文件是否为一个路径
        File file = getFile();
        return file.isDirectory();
    }
    public String toString() { //获取文件夹名称
        File file = getFile();
        String filename = file.toString();
        int index = filename.lastIndexOf("\\");
        return (index != -1 && index != filename.length() - 1) ? filename
            .substring(index + 1) : filename;
    }
    public String getPathString() { //获取文件路径名称
```



363





```
File file = getFile();
String filename = file.getAbsolutePath();
return filename;
}
public void explore() {
    if (!isDirectory()) {
        return;
    }
    if (!isExplored()) {
        File file = getFile();
        File[] children = file.listFiles();
        for (int i = 0; i < children.length; ++i) {
            if (children[i].isDirectory()) {
                //从其父节点移除 children, 并通过将其添加到此节点的子数组的结尾, 使其成为此节点的子节点
                add(new MyNode(children[i]));
            }
        }
        explored = true;
    }
}
}
```

完成在树组件中显示系统文件结构, 首先需要确定根节点, 以及各个子节点。本实例中将“桌面”设置为树结构的根节点。通过 `FileSystem` 类的 `getRoots()` 方法可获取系统的所有根分区, 在 Windows 中, 这将是“桌面”文件夹, 而在 DOS 中将是 A: 到 Z: 的驱动器。本模块将实例化 `JTree` 组件包装在 `getTree()` 方法中, 具体代码如下:

```
private JTree getTree() {
    File[] root = (new FileSystem()).getRoots(); //获取系统文件的根节点
    MyNode fileNode = new MyNode(root[0]);
    fileNode.explore();
    JTree jtree = new JTree(new DefaultTreeModel(fileNode));
    jtree.getSelectionModel().setSelectionMode(
        TreeSelectionMode.SINGLE_TREE_SELECTION);
    //设置树的选择模型为一次只选择一个路径
    JScrollPane sp = new JScrollPane(jtree);
    sp.setBorder(BorderFactory.createEtchedBorder(Color.white, new Color(
        148, 145, 140)));
    jtree.setShowsRootHandles(true); //如果在树的最高层显示句柄
    jtree.addTreeExpansionListener(new TreeExpansionListener() {
        //当树中某一项被折叠时调用
        public void treeCollapsed(TreeExpansionEvent e) {
        }
        public void treeExpanded(TreeExpansionEvent e) { //当树中某一项被展开时调用
            TreePath treepath = e.getPath(); //表示节点的路径
            获取路径中最后一个组件
            MyNode node = (MyNode) treepath.getLastPathComponent();
            if (!node.isExplored()) {
                DefaultTreeModel model = ((DefaultTreeModel) jtree
                    .getModel());
                node.explore();
                model.nodeStructureChanged(node); //更改各节点调用的方法
            }
        }
    });
    return jtree;
}
```







### 13.3.5 实现显示系统文件夹中的文件

当单击树结构中的某个子节点后,可将用户选择的某个地址中的文件显示在主窗体的表格中,效果如图 13.14 所示。

当单击树结构中的某一节点后,程序会将该节点所指示地址中的文件信息添加到表格中。完成这一功能,首先创建检索文件夹中的文件方法 `getFileList()`,该方法以 `String` 对象作为参数,在调用该方法时,可通过给定不同的参数将系统文件夹中的文件信息检索出来,具体代码如下:

文件名	大小	日期
12.mp3	112K	2008-5-21 9:36:25
15766.rar	2386K	2008-5-7 13:48:32
4.mp3	5807K	2008-5-21 9:36:34
52.mp3	5807K	2008-5-21 9:36:34
aa.wmv	23999K	2008-5-23 8:23:38
aaaaa.avi	43254K	2008-5-7 15:21:28
ad.avi	2K	2008-6-2 9:41:13
Aliree.avi	48493K	2008-5-23 8:23:40
blog.rar	2638K	2008-5-11 10:2
Encrypt.java	1K	2008-5-6 11:00:25
geg.wav	7054K	2008-5-21 14:2
jef-2_1_le-all	1959K	2008-5-7 14:59:24
jef-2_1_le-wind	5218K	2008-5-7 14:59:25

图 13.14 某一系统文件夹下的文件信息

```
public List getFileList(String strPath) {
    LinkedList filelist = new LinkedList();
    File dir = new File(strPath);
    File[] file = dir.listFiles();
    if((file != null)&&file.length > 0){
        for (int i = 0; i < file.length; i++) {
            if (file[i].isDirectory()) {
                getFileList(file[i].getAbsolutePath()); //调用本方法
            } else {
                filelist.add(file[i]); //向集合中添加元素
            }
        }
    }
    return filelist;
}
```

要完成当单击指定的节点时,将该节点所指示文件路径中的文件信息添加到表格中,应该在树组件中添加鼠标监听器,具体代码如下:

```
jtree.addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
        JTree tree = (JTree) e.getSource();
        int row = tree.getRowForLocation(e.getX(), e.getY()); //返回指定位置的行
        if (row == -1) {
            return;
        }
        TreePath path = tree.getPathForRow(row); //获取指定行路径
        if (path == null) {
            return;
        }
        MyNode node = (MyNode) path.getLastPathComponent();
        if (node == null) {
            return;
        }
        String filepath = node.getString();
        String newPath = filepath.replace("\\", "/"); //获取该节点所指定路径
        //将路径中的“\\”替换为“/”
        DefaultTableModel model = (DefaultTableModel) fileList
            .getModel(); //获取表格模型
        if (fileList.getRowCount() != 0) { //当表格中有数据时
            //将表格致空
            ((DefaultTableModel) fileList.getModel()).setRowCount(0);
        }
        list = fileHeald.getFileList(newPath); //获取该文件夹中的所有文件
    }
});
```



365





```
for (int i = 0; i < list.size(); i++) {
    MyFile myFile = new MyFile(list.get(i).toString()); //见说明
    File file = (File) list.get(i); //获取指定文件信息
    String length = file.length() + "B "; //获取文件长度信息
    if (file.length() > 1000 * 1000 * 1000) {
        length = file.length() / 1000000000 + "G ";
    }
    if (file.length() > 1000 * 1000) {
        length = file.length() / 1000000 + "M ";
    }
    if (file.length() > 1000) {
        length = file.length() / 1000 + "K ";
    }
    String modifDate = new Date(file.lastModified())
        .toLocaleString();
    if (!file.canRead()) {
        length = "未知";
        modifDate = "未知";
    }
    model.addRow(new Object[] {
        myFile.toString((File) list.get(i)), length,
        modifDate }); //将文件名、文件大小、文件创建日期添加到表格
}
});
```

在上述代码中，提到了继承自 File 类的 MyFile 类，该类的 toString() 方法以 File 对象作为参数，并将文件名称返回，具体代码如下：

```
import java.io.File;
public class MyFile extends File{
    public MyFile(String pathname) {
        super(pathname);
    }
    public String toString(File file){
        String fileStr = file.getAbsolutePath(); //返回此抽象路径名的绝对路径名字符串
        int index = fileStr.lastIndexOf("\\");
        String newiPath = null;
        if(index != -1){
            newiPath = fileStr.substring(index+1, fileStr.length()); //截取字符串
        }
        return newiPath;
    }
}
```

## 13.4 新建文件

### 13.4.1 功能概述

文件分割模块中添加了新建文件和文件夹的功能，用户可以通过单击“文件”菜单中的“新建文件”菜单项来完成新建文件功能。用户可以选择新建文件的地址、文件名、文件格式。“新建文件系列”运行如图 13.15 所示。

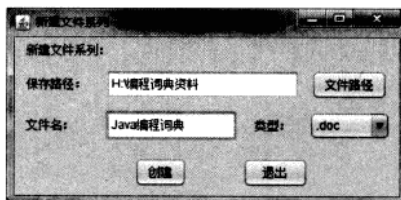


图 13.15 “新建文件系列”对话框



366







### 13.4.2 实现新建文件

当用户在“新建文件系列”对话框中指定完保存路径和文件名以后,单击“创建”按钮,将在指定位置完成文件的创建。首先编写新建文件的方法,具体代码如下:

```
public void createFile(String fileName) {    //该方法可通过抽象路径名创建文件
    try {
        String myFileName = fileName;
        File file = new File(myFileName);    //创建 File 对象
        if (!file.exists()) {                //如果该文件不存在
            file.createNewFile();            //新建文件
        }
    } catch (Exception e) {
        System.out.println("新建文件操作出错");
        e.printStackTrace();
    }
}
```

在新建文件窗体中,用户单击“文件路径”按钮后,可弹出“文件选择”对话框,用户可选择新建文件的保存地址,程序会将用户选择的地址添加到“文件路径”文本框中。“文件选择”对话框包装在 MyFileChooser 类中,读者可参考光盘中的源程序。“文件路径”按钮的单击事件具体代码如下:

```
private void filePathActionPerformed(java.awt.event.ActionEvent evt) {
    MyFileChooser myfile = new MyFileChooser(); //创建“文件选择”对话框实例
    myfile.getFolder();                          //调用选择文件路径方法
    String strPath = myfile.getFolderPath();    //获取用户选择的文件路径地址
    fileTextField.setText(strPath);             //设置“文件路径”文本框内容
}
```

当用户单击“创建”按钮后,会调用新建文件方法 createFile(),实现按指定路径创建文件,在“创建”按钮的单击事件中,首先判断用户选择新建文件的路径和文件名是否为空,其次,还需要判断在指定的路径中是否有与用户输入的文件名相同的文件。只有在用户确定了新建文件的地址和文件名,并且在该路径中不存在与用户输入的文件名相同的文件时,才可以完成新建文件。具体代码如下:

```
private void createJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //判断“文件路径”与“文件名”文本框是否为空
    if (fileTextField.getText().equals("") || fileName.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "文件名称或地址没有填写完整", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    //获取用户选择的文件路径与文件名、文件格式
    String strFile = fileTextField.getText() + "\\\" + fileName.getText()
        // jComboBox1 是文件格式下拉列表
        + jComboBox1.getSelectedItem().toString();
    File file = new File(strFile); //创建 File 对象
    if (file.exists()) {           //如果该文件存在
        JOptionPane.showMessageDialog(this, "文件已存在,请重新换一个文件名", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        fileName.setText("");      //将“文件名”文本框清空
        return;
    }
    FileHeald fileHeald = new FileHeald();
}
```





```
fileHeald.createFile(fileTextField.getText() + "\\\"
    + fileName.getText() + jComboBox1.getSelectedItem().toString());
//调用新建文件方法
JOptionPane.showMessageDialog(this, "文件创建成功", "信息对话框",
    JOptionPane.WARNING_MESSAGE);
}
```

### 13.4.3 实现新建文件夹

用户单击“新建”文件菜单中的“新建文件夹”菜单项，就会弹出“新建文件夹”对话框，用户可以选择新建文件夹的路径和文件夹名称，如图 13.16 所示。

首先应该编写新建文件夹方法，具体代码如下：

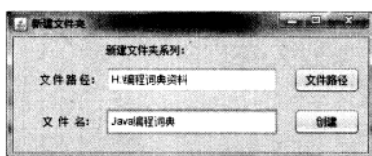


图 13.16 “新建文件夹”对话框

```
public void createFolder(String strPath) { //通过指定路径实现新建文件夹
    try {
        String filePath = strPath;
        File myFilePath = new File(filePath); //创建 File 对象
        if (!myFilePath.exists()) { //如果该对象不存在
            myFilePath.mkdir(); //创建此抽象路径名指定的目录
        }
    } catch (Exception e) {
        System.out.println("新建文件夹操作出错");
        e.printStackTrace();
    }
}
```

在“新建文件夹”对话框中，当用户单击“文件路径”按钮时，将会弹出“文件选择”对话框，并将“文件路径”文本框设置为用户选择的创建文件夹地址。“文件路径”按钮的单击事件具体代码如下：

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    MyFileChooser myfile = new MyFileChooser();
    myfile.getFolder(); //调用获取“选择文件夹”对话框
    String strPath = myfile.getFolderPath(); //获取用户选择的文件夹路径
    pathjTextField.setText(strPath); //设置“文件路径”文本框显示文本
}
```

用户单击“创建”按钮，就会调用新建文件夹的方法 `createFolder()`，完成新建文件夹的操作。在“创建”按钮的单击事件中，首先判断“文件路径”与“文件名”文本框是否为空，其次需要判断在用户指定的文件路径中，是否有与用户输入的文件夹名称相同的文件夹。只有在用户输入合法的文件地址和文件名后，才能完成新建文件夹。具体代码如下：

```
private void createJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //判断“文件路径”与“文件名”文本框是否为空
    if (pathjTextField1.getText().equals("") || pathjTextField.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "文件名称或地址没有填写完整", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return; //退出程序，不执行以下代码
    }
}
```







```

}
//获取文件路径与文件夹名称
String strFile = pathjTextField.getText()+"\\"+pathjTextField1.getText();
File file = new File(strFile); //创建 File 对象
if(file.exists()){ //如果该文件夹存在
    JOptionPane.showMessageDialog(this, "文件夹已存在, 请重新换一个文件名", "信息对话框", JOptionPane.WARNING_MESSAGE);
    pathjTextField1.setText(""); //将“文件夹名称”文本框清空
    return; //退出程序
}
FileHeald fileHeald = new FileHeald();
//根据用户选择的文件地址与文件名称创建文件夹
fileHeald.createFolder(pathjTextField.getText()+"\\"+pathjTextField1.getText());
JOptionPane.showMessageDialog(this, "文件夹创建成功", "信息对话框",
JOptionPane.WARNING_MESSAGE);
}

```

## 13.5 实现文件搜索

### 13.5.1 功能概述

为了使用户对系统盘符中的文件更加了解, 本模块添加了文件搜索功能。文件搜索是按照指定的文件格式进行搜索, 文件搜索界面如图 13.17 所示。

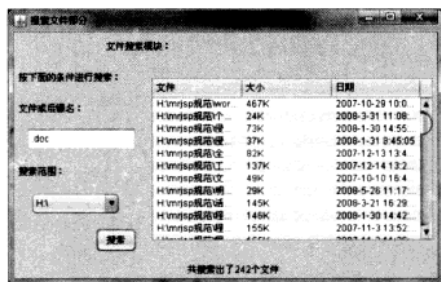


图 13.17 文件搜索界面

### 13.5.2 在下拉列表中添加有效盘符

在“搜索文件”窗体的“搜索范围”下拉列表中显示了本地有效盘符, 要想将本地有效盘符在窗体中显示, 需要将获取的本地有效盘符添加到窗体的下拉列表中, 具体代码如下:

```

boundLabel.setText("搜索范围: "); //设置文本提示信息
FileHeald fileHeald = new FileHeald();
java.io.File[] files = fileHeald.getRoot(); //调用获取本地有效磁盘方法
String str[] = new String[files.length]; //创建字符串数组
for(int i = 0;i<files.length;i++){
    str[i] = files[i].getPath();
}
//将获取的磁盘信息添加到下拉列表中
jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(str));

```





### 13.5.3 实现文件搜索功能

当用户单击“搜索”按钮后，程序将在指定的盘符中搜索文件，将文件信息显示在表格中，并给出文件的总数。首先编写检索文件夹（包括子文件夹）中文件的方法，具体代码如下：

```
public List getList(String path){           //通过给定参数，可获取文件夹中的文件集合
    LinkedList<File> list=new LinkedList<File>(); //创建集合对象，用于保存文件夹
    ArrayList<String> listPath = new ArrayList<String>(); //用于保存文件名
    File dir=new File(path);
    File file[]=dir.listFiles();           //获取抽象路径名数组
    for(int i=0;i<file.length;i++){        //循环遍历数组
        if(file[i].isDirectory())          //如果数组中的元素为路径名
            list.add(file[i]);             //向集合中添加元素
        else{
            listPath.add(file[i].getAbsolutePath()); //向文件名集合中添加元素
        }
    }
    File tmp;
    while(!list.isEmpty()){                //如果文件路径集合不为空，则进入集合
        tmp=list.removeFirst();            //移除并返回集合中的第一项
        if(tmp.isDirectory()){
            file=tmp.listFiles();
            if(file==null)continue;
            for(int i=0;i<file.length;i++){
                if(file[i].isDirectory())    //如果文件名集合中包含文件夹名
                    list.add(file[i]);        //向文件夹名集合中添加元素
            }
            listPath.add(file[i].getAbsolutePath());
        }
    }
    return listPath;                       //返回集合对象
}
```



moveFirst()方法用来移除并返回集合中的第一项，在上段代码中，由于进入循环的条件为保存文件夹名称的集合不为空。此时，如果使用 getFirst()方法，获取集合中的第一项，可能导致无法退出 while 循环。

用户在确定要搜索的文件扩展名后，选择要进行搜索的地址，单击“搜索”按钮后，程序将通过线程搜索文件，并将搜索到的文件信息显示在表格中。“搜索”按钮的单击事件具体代码如下：

```
private void searchButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (jTable1.getRowCount() != 0) {      //当表格不为空时，进行清空表格
        ((DefaultTableModel) jTable1.getModel()).setRowCount(0);
    }
    if(nameTextField.getText().equals("")){ //如果“文件名称”对话框为空
        JOptionPane.showMessageDialog(this,"没有指定完整的搜索条件","确认对话框",
        JOptionPane.WARNING_MESSAGE);
    }
    return;                                //退出程序，不执行一下代码
}
```







```

}
else{
    Thread thread = new Thread(this);
    thread.start();
    countLabel.setText("文件搜索中...");
}
}

```

完成线程真正功能的代码放在线程的 `run()` 方法中。当调用开启线程方法 `start()` 时将执行线程，也就是调用 `run()` 方法，`run()` 方法的具体代码如下：

```

public void run() {
    DefaultTableModel model = (DefaultTableModel) jTable1.getModel(); //获取表模型
    FileHeald fileHeald = new FileHeald();
    java.util.List list = fileHeald.getList(jComboBox1.getSelectedItem().toString());
    //jComboBox1 为保存系统有效盘符的下拉列表
    //nameTextField 为用户要搜索的文件扩展名文本框
    String strName = nameTextField.getText();
    if(strName.indexOf(".") == -1){ //如果用户填写的文件后缀名不包含 "."
        strName = "." + strName;
    }
    if(!list.isEmpty() && list.size() > 0){
        for(int i = 0; i < list.size(); i++){
            String path = list.get(i).toString();
            //如果集合中的元素后缀与用户要进行搜索的文件后缀相同
            if((path.endsWith(strName))){
                File file = new File(list.get(i).toString());
                length = file.length() + "B ";
                if (file.length() > 1000 * 1000 * 1000) {
                    length = file.length() / 1000000000 + "G ";
                }
                if (file.length() > 1000 * 1000) {
                    length = file.length() / 1000000 + "M ";
                }
                if (file.length() > 1000) {
                    length = file.length() / 1000 + "K ";
                }
                modifDate = new Date(file.lastModified())
                    .toLocaleString();
                if (!file.canRead()) {
                    length = "未知";
                    modifDate = "未知";
                }
                strFileName = list.get(i).toString();
                model.addRow(new Object[] {
                    strFileName, length,
                    modifDate }); //向数据表添加数据
            }
        }
        countLabel.setText("共搜索出了" + jTable1.getRowCount() + "个文件");
    }
}
}

```

## 13.6 实现批量复制



171



### 13.6.1 功能概述

通过文件分割模块，可快速实现批量复制。为了方便用户操作，在批量复制子模块



中设计了按照文件格式批量进行复制，以及复制整个文件夹。

## 13.6.2 实现复制指定文件

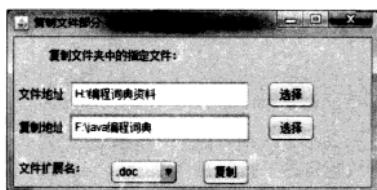


图 13.18 批量复制指定文件

用户单击主窗体中的“编辑”菜单/“批量复制”/“复制指定文件”菜单项，可完成批量复制指定的文件。通过批量复制指定文件子模块，可以将用户选择文件夹中指定格式的文件，全部复制到相应文件夹中，批量复制指定文件子模块的运行结果如图 13.18 所示。

完成批量复制指定文件，首先需要获取用户选择的文件夹中的文件集合，如果集合中的元素与用户要进行复制的扩展名相同，则调用复制文件方法，完成文件的

复制。完成文件复制方法的具体代码如下：

```
public void copyFile(String oldPath, String newPath) {  
    //该方法以复制文件的路径、复制后文件的路径作为参数  
    try {  
        int bytesum = 0;  
        int byteread = 0;  
        File oldfile = new File(oldPath);  
        if (oldfile.exists()) { // 文件存在时  
            InputStream inStream = new FileInputStream(oldPath); // 读入原文件  
            FileOutputStream fs = new FileOutputStream(newPath);  
            byte[] buffer = new byte[1444];  
            while ((byteread = inStream.read(buffer)) != -1) {  
                bytesum += byteread; // 获取文件大小  
                fs.write(buffer, 0, byteread);  
            }  
            inStream.close();  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

当用户单击复制窗体中的“复制”按钮后，可完成文件的复制。在“复制”按钮的单击事件中，首先判断用户是否输入完整的文件地址、复制地址。由于用户指定的文件夹中可能会有很多符合复制条件的文件，完成复制可能会需要一些时间，为了达到界面友好的效果，需要给用户提供相应的提示信息，具体代码如下：

```
private void copyJButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    //判断用户是否输入完整的文件地址与保存地址  
    if ((FileTextField.getText().equals("")) || (FileSaveTextField.getText().  
        equals(""))){  
        JOptionPane.showMessageDialog(this, "没有将文件地址复制地址添加写完整", "信息对话",  
            JOptionPane.WARNING_MESSAGE);  
        return;  
    }  
    //创建 FileHeald 类对象，该类包装各种文件操作方法  
    FileHeald fileHeald = new FileHeald();  
    //获取文件集合  
    java.util.List list = fileHeald.getList(FileTextField.getText());  
}
```







```

if(!list.isEmpty()){
    MyHint hint = new MyHint();                //创建提示窗体对象
    hint.setVisible(true);
    for(int i = 0 ;i<list.size();i++){
        String strFileName = list.get(i).toString(); //获取集合中的元素
        if(strFileName.endsWith(jComboBox1.getSelectedItem().toString())){
            // jComboBox1 为用户提供选择复制文件格式的下拉列表
            int index = strFileName.lastIndexOf("\\");
            String strNewName = strFileName.substring(index+1,strFileName.
            length());
            fileHeald.copyFile(strFileName, FileSaveTextField.getText()+
            "\\\"+strNewName);
            //调用文件复制方法
        }
    }
    hint.setVisible(false);
    JOptionPane.showMessageDialog(this, "文件复制完毕", "消息对话框",JOptionPane.
    WARNING_MESSAGE);
}
if(list.isEmpty()){
    JOptionPane.showMessageDialog(this, "文件夹为空", "信息对话框",JOptionPane.
    WARNING_MESSAGE );
}
}

```

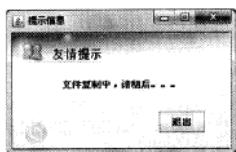


图 13.19 “提示信息”对话框

在上段代码中，当程序正在复制文件时，会调用 `MyHint` 类，提示用户正在复制文件。需要注意的是，如果复制文件需要的时间很短，则可能不会显示“提示信息”对话框，“提示信息”对话框的运行结果如图 13.19 所示。

### 13.6.3 实现复制整个文件夹

用户依次单击“编辑”菜单下的“批量复制”/“复制整个文件夹”菜单项，可实现将用户选择的文件夹复制到指定位置。复制文件夹窗体的运行结果如图 13.20 所示。

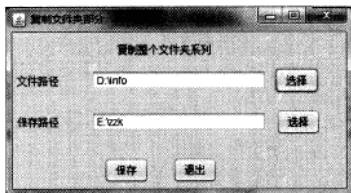


图 13.20 复制文件夹窗体

要完成复制整个文件夹内容，首先，需要编写复制整个文件夹的方法 `copyFolder()`，该方法有两个 `String` 类型的参数，分别代表要进行复制的文件夹地址和复制文件的保存地址。通过调用 `copyFolder()` 方法可以实现复制该文件夹及其子文件夹下的所有文件。具体代码如下：







```
public void copyFolder(String oldPath, String newPath) {
    try {
        (new File(newPath)).mkdirs(); //如果文件夹不存在, 则建立新文件夹
        File oldfile = new File(oldPath);
        String[] file = oldfile.list(); //获取抽象路径名表示的目录中的文件和目录
        File temp = null;
        for (int i = 0; i < file.length; i++) { //循环遍历数组
            if (oldPath.endsWith(File.separator)) { //如果要进行复制的文件夹是以系统默认的文件分隔符结尾的
                temp = new File(oldPath + file[i]); //实例化文件对象
            } else {
                temp = new File(oldPath + File.separator + file[i]);
            }
            if (temp.isFile()) { //如果 temp 对象是一个文件对象
                FileInputStream input = new FileInputStream(temp);
                FileOutputStream output = new FileOutputStream(newPath
                    + "/" + (temp.getName()).toString());
                byte[] b = new byte[1024 * 5];
                int len;
                while ((len = input.read(b)) != -1) {
                    output.write(b, 0, len); //循环写入文件信息
                }
                output.flush();
                output.close();
                input.close();
            }
            if (temp.isDirectory()) { //如果是子文件夹
                //再次调用本方法
                copyFolder(oldPath + "/" + file[i], newPath + "/" + file[i]);
            }
        }
    } catch (Exception e) {
        System.out.println("复制整个文件夹内容操作出错");
        e.printStackTrace();
    }
}
```

当用户单击复制文件夹窗体中的“保存”按钮后, 程序将调用复制文件夹的方法。由于用户要进行复制的文件夹可能会有很多的文件, 为了避免在复制大量文件时可能会出现死机现象, 可以在单独的线程中完成文件夹的复制。在“保存”按钮的单击事件中, 完成复制文件地址的检测及启动线程等。具体代码如下:

```
private void copyButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //判断用户是否确定进行复制的文件夹与文件夹的保存地址为空
    if (pathTextField.getText().equals("") || (pathTextField1.getText().equals(""))){
        JOptionPane.showMessageDialog(this, "请将信息添加完整", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    String strPath = pathTextField.getText();
    int index = strPath.indexOf("\\");
    folderStr = strPath.substring(index, strPath.length());
    if (index == strPath.length() - 1) { //判断用户是否选择合法的文件夹
        int n = JOptionPane.showConfirmDialog(this, "请选择指定文件夹", "确认对话",
            JOptionPane.YES_NO_OPTION);
        if (n == JOptionPane.YES_OPTION) {
            pathTextField.setText("");
            pathTextField.requestFocus();
            return;
        }
    }
}
```







```

    }
    if(n == JOptionPane.NO_OPTION){
        return;
    }
}
else{
    Thread thread = new Thread(this);
    thread.start();           //启动线程
}
}

```

在线程的 `run()` 方法中调用复制文件夹的方法 `copyFolder()`，并指定复制文件夹的路径和复制后的保存路径。为了区分，程序将复制后的文件名称设置为原文件夹名称后加“Copy”。具体代码如下：

```

public void run() {
    MyHint myHint = new MyHint();
    myHint.setVisible(true);           //显示提示窗体
    myHint.setBounds(200, 250, 300, 200);
    FileHeald fileHeald = new FileHeald(); //创建包装各种文件操作类对象
    fileHeald.copyFolder(pathTextField.getText(), pathTextField1.getText()+
        "\\ "+folderStr+"Copy");
    //调用复制文件夹方法
    myHint.setVisible(false);          //不显示提示窗体
    int t = JOptionPane.showConfirmDialog(this, "复制完毕，需要继续吗", "确认对话框",
        JOptionPane.YES_NO_OPTION);
    if(t == JOptionPane.YES_OPTION){
        pathTextField.setText("");      //将“文件路径”文本框清空
        pathTextField1.setText("");    //将“保存路径”文本框清空
    }
    if(t == JOptionPane.NO_OPTION){
        dispose();                     //释放窗体占用的资源
    }
}

```

## 13.7 实现文件批量移动

### 13.7.1 功能概述

为了方便文件和文件夹的移动，本模块设计了文件批量移动子模块，主要实现文件的批量移动及移动整个文件夹的功能。文件移动是指将文件从原地址移动到另一个地址，也就是说文件夹移动是指将整个文件夹从原地址移动到目标地址。

### 13.7.2 实现移动指定文件

单击主窗体上的“编辑”菜单，选择“批量移动”/“移动指定文件”菜单项，可实现批量移动文件的功能。“移动指定文件”对话框如图 13.21 所示。

要完成移动文件，需要将用户要移动的文件复制

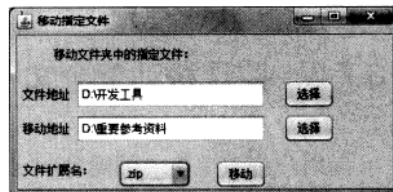


图 13.21 “移动指定文件”对话框





到相应地址，再将原地址中的文件删除。当用户单击“移动”按钮后，即可完成文件移动。由于要移动的文件可能会有很多，为了避免在文件移动过程中出现死机现象，程序将在单独的线程中完成文件的移动。“移动”按钮的单击事件中完成验证“文件地址”、“移动地址”文本框是否为空，并启动线程。具体代码如下：

```
private void copyJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //验证“文件地址”、“保存地址”文本框是否为空
    if((FileTextField.getText().equals("")) || (FileSaveTextField.getText().equals(""))){
        JOptionPane.showMessageDialog(this, "没有将文件地址添◆◆写完◆", "信息对话◆",
        JOptionPane.WARNING_MESSAGE);
        return;
    }
    Thread thread = new Thread(this);
    thread.start();           //启动线程
}
```

当调用线程的 `start()` 方法开启线程后，会调用线程的 `run()` 方法。在 `run()` 方法中实现文件移动，具体代码如下：

```
public void run() {
    FileHeald fileHeald = new FileHeald();           //创建对文件操作类对象
    //调用获取文件夹中所有文件的方法
    java.util.List list = fileHeald.getList(FileTextField.getText());
    if(!list.isEmpty()){
        MyRemoveMessage myRemoveMessage = new MyRemoveMessage();//该类为提示信息类
        myRemoveMessage.setVisible(true);
        for(int i = 0 ; i<list.size(); i++){
            String strFileName = list.get(i).toString();           //获取集合中元素
            if(strFileName.endsWith(jComboBox1.getSelectedItem().toString())){
                // jComboBox1 为“文件扩展名”下拉列表
                int index = strFileName.lastIndexOf("\\");
                String strNewName = strFileName.substring(index+1,
                strFileName.length());
                fileHeald.copyFile(strFileName, FileSaveTextField.getText()+
                "\\ "+strNewName);
                //调用复制文件方法
                File file = new File(strFileName);
                file.delete();           //将文件删除
            }
        }
        myRemoveMessage.setVisible(false);
        JOptionPane.showMessageDialog(this, "文件移动完毕", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
    }
    if(list.isEmpty()){           //当文件集合为空时，给出提示信息
        JOptionPane.showMessageDialog(this, "文件夹为空", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
    }
}
```



### 13.7.3 实现移动整个文件夹



当用户依次单击“编辑”/“批量移动”/“移动文件夹”菜单项，程序将根据用户选择的“文件地址”与“移动地址”完成移动整个文件夹的操作。移动文件夹窗体的运行效果如图 13.22 所示。



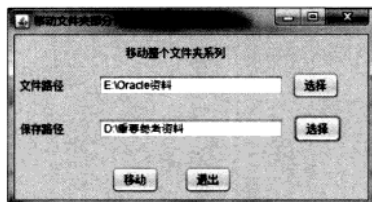


图 13.22 移动文件夹

实现移动整个文件夹内容, 需要将要进行移动的文件夹复制到用户输入的“保存路径”文本框指示的地址中, 再将原文件夹删除。在“移动”按钮的单击事件中, 完成验证“文件路径”、“保存路径”文本框中的文件地址是否合法, 并启动新线程。具体代码如下:

```
private void copyButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String strPath = pathTextField.getText(); //获取“文件地址”文本框中的信息
    String strPathSave = pathTextField1.getText(); //获取“保存路径”文本框中的信息
    if(strPath.equals("") || strPathSave.equals("")){
        JOptionPane.showMessageDialog(this, "请输入完整的文件地址", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    int index = strPath.indexOf("\\");
    folderStr = strPath.substring(index, strPath.length());
    if(index == strPath.length()-1){ //验证用户输入的文件地址是否合法
        int n = JOptionPane.showConfirmDialog(this, "请选择指定文件夹", "确认对话框",
            JOptionPane.YES_NO_OPTION);
        if(n == JOptionPane.YES_OPTION){
            pathTextField.setText("");
            pathTextField.requestFocus(); //获取焦点
            return;
        }
        if(n == JOptionPane.NO_OPTION){
            return;
        }
    }
    else{
        Thread thread = new Thread(this);
        thread.start(); //启动线程
    }
}
```

在线程的 run() 方法中实现复制文件夹, 并将原地址中的文件夹删除。线程 run() 方法的具体代码如下:

```
public void run() {
    FileHeald fileHeald = new FileHeald(); //创建文件操作类对象
    //创建文件提示信息窗体对象
    MyRemoveMessage removeMessage = new MyRemoveMessage();
    removeMessage.setVisible(true); //设置窗体显示
    fileHeald.copyFolder(pathTextField.getText(), pathTextField1.getText()+
        "\\ "+folderStr);
    //调用复制整个文件夹方法
    fileHeald.deleteDirs(new File(pathTextField.getText()));
    //将原地址中的文件夹删除
    removeMessage.setVisible(false);
}
```





```
int t = JOptionPane.showConfirmDialog(this, "移动完毕, 需要继续吗?", "确认对话",
JOptionPane.YES_NO_OPTION);
if(t == JOptionPane.YES_OPTION){
    pathTextField.setText("");
    pathTextField1.setText("");
}
if(t == JOptionPane.NO_OPTION){
    dispose();
}
}
```

## 13.8 实现批量删除

### 13.8.1 功能概述

文件分割模块中提供了批量删除功能, 通过该子模块可以快速地删除文件。在批量删除子模块中可以根据文件扩展名和最后一次修改时间进行删除。“批量删除模块”对话框如图 13.23 所示。

在“批量删除模块”对话框中, 可以设置要删除文件的扩展名、文件最后一次修改的日期, 并将满足条件的文件信息显示在窗体下方的文件列表中。

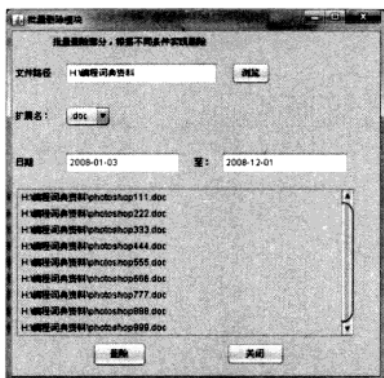


图 13.23 “批量删除模块”对话框

### 13.8.2 “扩展名”下拉列表设计

在“扩展名”下拉列表中包含常见的文件格式, 用于方便用户指定要删除的文件格式。在“扩展名”下拉列表中添加了 `ItemListener` 监听器, 当“扩展名”下拉列表中的值发生变化时, 会调用 `itemStateChanged()` 方法。为列表添加监听器的具体代码如下:

```
jComboBox1.setModel(new javax.swing.DefaultComboBoxModel
    (new String[] { " ", ".doc", ".mp3", ".txt", ".avi", ".exe",
        ".zip", ".rar", ".JPG" }));
//设置下拉列表的文本内容
jComboBox1.addItemListener(new java.awt.event.ItemListener() { //为列表添加监听器
    public void itemStateChanged(ItemEvent e) {
        selectItem(e); //调用 selectItem() 方法
    }
});
```







在上述代码中当“扩展名”下拉列表中的数据发生变化时,程序将调用 selectItem() 方法,并在该方法中实现了向文件列表中添加数据的功能,具体代码如下:

```
private void selectItem(java.awt.event.ItemEvent evt) {
    FileHeald fileHeald = new FileHeald(); //调用封装各种文件操作方法
    if (!pathTextField.getText().equals("")) { //如果“文件路径”文本框不为空
        //获取指定文件夹中文件集合
        List list = fileHeald.getFileList(pathTextField.getText());
        if (!list.isEmpty() && list.size() > 0) { //如果集合不为空
            fileName.removeAllElements(); //将保存文件集合的 Vector 对象清空
            jList1.removeAll(); //将显示文件集合列表清空
            jList1.revalidate(); //刷新列表
            for (int i = 0; i < list.size(); i++) {
                File str = (File) list.get(i);
                String sPath = str.getAbsolutePath();
                if (sPath.endsWith(jComboBox1.getSelectedItem().toString())) {
                    addList(str.getAbsolutePath()); //将满足条件的文件信息添加到列表中
                }
            }
        } else {
            fileName.removeAllElements(); //如果没有满足条件的添加文件
            jList1.removeAll(); //将列表清空
            jList1.revalidate();
            JOptionPane.showMessageDialog(this, "没有符合条件的文件", "信息对话框",
                JOptionPane.ERROR_MESSAGE); //给出提示信息
        }
    }
}
```

### 13.8.3 文件日期文本框设计

在文件分割窗体中,设计了“文件日期”文本框。用户可以通过文本框中的日期,将最后一次修改日期在用户给定的两个日期之间的文件信息添加到文件信息列表中。

#### 1. 编写比较时间前后的方法

文件列表中显示的文件信息,是最后一次修改时间在“文件日期”文本框中两个日期之间的文件信息。因此,需要编写比较两个日期前后的方法,具体代码如下:

```
public boolean isDateBefore(String date1, String date) {
    boolean b = true; // 根据该方法的返回值设置变量
    // 获得时间格式,为系统默认的格式
    DateFormat df = DateFormat.getDateInstance();
    try {
        b = df.parse(date1).before(df.parse(date)); // 判断 date1 是否在 date2 之前
    } catch (ParseException e) {
        e.printStackTrace();
    }
    return b; //如果 date1 在 date2 之前则返回 true
}
```



DateFormat 类是日期/时间格式化子类的抽象类,它以与语言无关的方式格式化并解析日期或时间。该类的 getDateInstance() 方法可获取日期格式器,该格式器具有





默认语言环境的默认格式化风格。该类的 `parse()` 方法可从给定字符串的开始解析文本，以生成一个日期，`parse()` 方法返回 `Date` 对象。通过调用 `Date` 类的 `before()` 来测试此日期是否在指定日期之前。

## 2. 设计文件日期文本框

用户在文件日期文本框中可输入标准时间，程序将按照用户输入的时间，与文件最后一次修改的时间相比较。通过在文件日期文本框中添加焦点事件监听器，实现当文本框失去焦点时判断用户是否输入了合法的日期，具体代码如下：

```
//为文本框添加焦点事件
fromTextField.addFocusListener(new java.awt.event.FocusListener(){
    public void focusGained(FocusEvent e) {
    }
    public void focusLost(FocusEvent e) {
        fouseLisenter(e); //文本框失去焦点处理事件
        //调用 fouseLisenter() 方法
    }
});
private void fouseLisenter(java.awt.event.FocusEvent fvt){
    String fromstr = fromTextField.getText(); //获取文本框中的数据信息
    String fromt = "\\d{4}-\\d{2}-\\d{2}"; //正则表达式，用于判断文本框中信息是否合法
    //判断文本框中的字符串是否与正则表达式相匹配
    if(!fromstr.matches(fromt) && !(fromTextField.getText().equals(""))){
        int messageindex = JOptionPane.showConfirmDialog(this,
            "日期格式为: '2001-12-12'", "确定对话", JOptionPane.YES_NO_OPTION);
        if(messageindex == JOptionPane.YES_OPTION){
            fromTextField.setText("");
            fromTextField.requestFocus(); //获取焦点
        }
        else if(messageindex == JOptionPane.NO_OPTION){
        }
    }
    else if(!pathTextField.getText().equals("") && !(fromTextField.getText().equals("")) && !(toTextField.getText().equals(""))){ //如果两个文件日期文本框都不为空
        FileHeald fileHeald = new FileHeald(); //创建操作文件对象
        //调用搜索文件夹中的文件方法
        List list = fileHeald.getFileList(pathTextField.getText());
        if(!list.isEmpty() && list.size() > 0){
            fileName.removeAllElements(); //将文件列表中的信息清空
            jList1.removeAll();
            jList1.revalidate();
            for(int i = 0 ; i < list.size(); i++){
                File str = (File)list.get(i); //获取文件集合中的元素
                //获取文件最后一次修改日期
                String modifDate = new Date(str.lastModified())
                    .toLocaleString();
                boolean boolfrom = fileHeald.isDateBefore(modifDate, toTextField.getText() + " 00:00:00");
                //调用判断时间先后方法
                boolean boolto = fileHeald.isDateBefore(fromTextField.getText() + " 00:00:00", modifDate);
                String sPath = str.getAbsolutePath(); //获取集合元素中的文件完整路径
                //如果文件日期在用户指定的日期之间
                if(boolfrom == true && boolto == true ){
                    jList1.removeAll(); //首先将文件信息列表清空
                    addList(str.getAbsolutePath()); //实现向文件列表添加数据
                }
            }
        }
    }
}
```







```

    }
}
else{
    fileName.removeAllElements();           //清空文件信息列表
    jList1.removeAll();
    jList1.revalidate();
}
}
}

```



上述代码中用于验证文本框中的数据是否是合法日期，使用了正则表达式，正则表达式通常用于判断语句中。正则表达式是含有一些具有特殊意义字符的字符串。例如，“\d”表示0~9中任何一个数字。正则表达式中允许使用限定符来限定元字符出现的次数。例如，“\d{4}”表示可以出现4次0~9的数字。

在上述代码中，实现向文件信息列表添加数据时，调用了 addList()方法，该方法的具体代码如下：

```

private void addList(String vf) {
    fileName.addElement(vf);           //向保存文件信息的文本框中添加信息
    jList1.setListData(fileName);      //向列表中添加 Vector 对象
}

```

用户单击“删除”按钮后，程序将对文件信息列表中列出的文件执行删除操作。“删除”按钮的事件代码如下：

```

private void deleteActionPerformed(java.awt.event.ActionEvent evt) {
    int messageindex = JOptionPane.showConfirmDialog(this, "确定要删除文件吗? ◆",
    "确定对话", JOptionPane.YES_NO_OPTION);
    if(messageindex == JOptionPane.YES_OPTION){
        if(fileName.size() > 0){           //fileName 为保存文件信息的 Vector 对象
            for(int i = 0;i<fileName.size();i++){
                //获取向量 Vector 对象中的元素
                String str = fileName.get(i).toString();
                File file = new File(str);
                file.delete();              //删除 File 对象
            }
            validate();
        }
    }
    else if(messageindex == JOptionPane.NO_OPTION ){
    }
}
}

```

## 13.9 实现批量重命名



381



### 13.9.1 功能概述

Windows 操作系统可以实现重命名文件操作，却不能实现批量重命名。在文件分割模

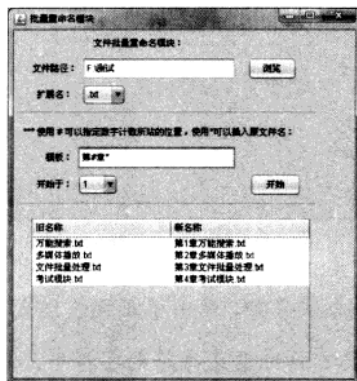


图 13.24 “批量重命名模块”对话框

块中添加了批量重命名功能,可以将一个文件夹内同一类型的文件按照一定的规则批量重命名。用户可以给出重命名模板,程序将根据模板对相应的文件进行重命名。除此之外还可以在重命名模板中添加特殊符号,程序会将这些特殊符号替换成原文件名或在重命名文件后需要进行编号的位置。“批量重命名模块”对话框如图 13.24 所示。

### 13.9.2 批量重命名文件

通过批量重命名子模块实现的是对原文件进行重命名,在调用文件重命名方法时,需要给定文件命名前的文件名和文件命名后的文件名。在重命名模板中,可以使用“#”来代替,重命名文件后的数字计数编号,并且可以在重命名模板中添加“\*”来代替原文件名。在重命名窗体中还设计了“开始于”下拉列表,用来表示重命名文件的数字计数的开始数。当用户确定了重命名模板后,单击“开始”按钮后,程序将进行重命名操作,“开始”按钮的单击事件代码如下:

```
private void beginJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // fileNameModel 为重命名模板文本框
    if (fileNameModel.getText().equals("")) {
        JOptionPane.showMessageDialog(this, "请确定重命名模板", "信息对话框",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    if (jTable1.getRowCount() != 0) { // jTable1 为显示新文件名和旧文件名的表格
        ((DefaultTableModel) jTable1.getModel()).setRowCount(0); // 将表格清空
    }
    FileHeald fileheald = new FileHeald(); // 创建对文件进行操作的类对象
    DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
    // 调用检索文件夹中所有文件的方法
    java.util.List list = fileheald.getFileList(pathTextField.getText());
    int bi = 0;
    if (!list.isEmpty() && (list.size() > 0)) { // 如果文件夹中的文件集合不为空
        for (int i = 0; i < list.size(); i++) { // 循环遍历集合
            java.io.File file = (java.io.File) list.get(i); // 获取集合对象
            if (file.getAbsolutePath().toString().endsWith(
                // jComboBox3 为文件后缀名
                jComboBox3.getSelectedItem().toString())) {
                // 获取文件对象的完整路径信息
                String strPath = file.getAbsolutePath();
                // 获取文件路径中最后一个“\\”处的索引
                int index = strPath.lastIndexOf("\\");
                String newStr = strPath.substring(index + 1, strPath
                    .length()); // 截取字符串
            }
        }
    }
}
```







```
//获取字符串中最后一个“.”处的索引
int index2 = newStr.lastIndexOf(".");
String newstring = newStr.substring(0, index2); //截取字符串
int ind = strPath.lastIndexOf(".");
String pathn = strPath.substring(ind, strPath.length());
//获取用户给出的重命名模板
String strModel = fileNameModel.getText();
String newModel = strModel.replace("#",
    ((Integer.parseInt(jComboBox2.getSelectedItem()
        .toString())) + bi)
    + ""); //将模板中的“#”替换成用户选择的开始计数的数字
//将“*”替换为原文件名
String model2 = newModel.replace("*", newstring);
bi = bi + 1; //将用户选择的开始计数进行加 1 处理
String path = strPath.substring(0, index);
fileheald.renamePath(strPath, path + "\\\" + model2 + pathn);
//调用重命名方法
System.out.println("MODEL " + path + "\\\" + newModel
    + pathn);
model.addRow(new Object[] { newStr, model2 + pathn });
//将新文件名、旧文件名添加到表格中
    }
} else {
    JOptionPane.showMessageDialog(this, "没有符合条件的文件", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
}
}
```

在上段代码中多处应用了字符串处理方法,例如,获取某个字符的索引、字符串替换、字符串截取等,下面是本实例中用到的字符串处理方法及其语法说明。

#### 1) lastIndexOf()方法

语法:

```
lastIndexOf(String str)
```

说明: 返回指定子字符串在此字符串中最右边出现处的索引。

#### 2) substring()方法

语法:

```
substring(int beginIndex, int endIndex)
```

说明: 返回一个新字符串,它是此字符串的一个子字符串。该子字符串从指定的 beginIndex 处开始,直到索引 endIndex - 1 处的字符,因此,该子字符串的长度为 endIndex-beginIndex。

#### 3) replace()方法

语法:

```
replace(CharSequence target, CharSequence replacement)
```

说明: 使用指定的字面值替换序列替换此字符串所有匹配字面值目标序列的子字符串,该替换从字符串的开头向末尾执行。





## 13.10 批量修改文件编码格式

### 13.10.1 功能概述

为了方便跨平台操作，文件分割模块中添加了批量修改文件编码格式的功能，例如，编码格式为“UTF-8”的文件，使用文件编码格式为“GBK”的文件编辑器打开时，会出现中文乱码现象，这时，可以通过转变文件编码来解决中文乱码的问题。虽然，通过文件分割模块实现文件编码格式的转换没有像“批量重命名”子模块那样，在原文件的基础上进行操作。但为了保证数据安全，最好对原文件进行备份。批量修改文件编码界面运行结果如图 13.25 所示。

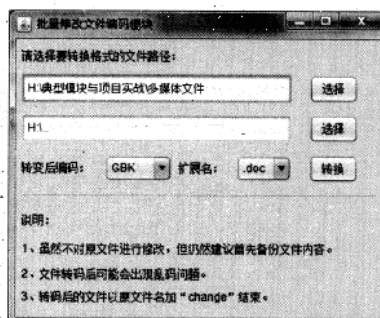


图 13.25 批量修改文件编码

### 13.10.2 批量修改文件编码

在“批量修改文件编码模块”对话框中，用户可选择要进行批量转换文件编码的文件地址，以转换后的文件保存地址，当用户单击“选择”按钮后，程序将弹出“文件选择”对话框，并将用户选择的地址添加到“文件路径”文本框中。“选择”按钮的单击事件具体代码如下：

```
private void fileButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    MyFileChooser myfile = new MyFileChooser(); //创建文件选择器对象  
    myfile.getFolder(); //该文件选择器只选择文件夹  
    String strPath = myfile.getFolderPath(); //获取用户选择的文件夹地址  
    //将用户选择的文件地址设置为文件框的显示信息  
    filePathTextField.setText(strPath);  
}
```

当用户单击“转换”按钮后，程序首先会检索用户选择的文件地址是否合法，然后调用文件编码格式转换方法，具体代码如下：

```
private void okjButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    //判断用户是否选择要进行文件转换的文件地址  
    if (filePathTextField.getText().equals("")) {  
        int n = JOptionPane.showConfirmDialog(this, "没有指定文件路径", "确认对话框",  
            JOptionPane.YES_NO_OPTION);  
        if (n == JOptionPane.YES_OPTION) {  
            filePathTextField.requestFocus(); //“文件路径”文本框获取焦点  
            return;  
        }  
        if (n == JOptionPane.NO_OPTION) {  
            return;  
        }  
    }  
    //判断用户是否选择了文件编码格式转换后的文件保存地址  
    if (fileSaveTextField.getText().equals("")) {
```







```
int n = JOptionPane.showConfirmDialog(this, "没有指定文件保存路径", "确认对话框",
    JOptionPane.YES_OPTION);
if (n == JOptionPane.YES_OPTION) {
    filePathTextField.requestFocus();
    return;
}
if (n == JOptionPane.NO_OPTION) {
    return;
}
}
FileHeald fileheald = new FileHeald(); //创建文件操作类对象
java.util.List list = fileheald
    .getFileList(filePathTextField.getText()); //检索文件夹中的所有文件
int bi = 1;
if (!list.isEmpty() && (list.size() > 0)) { //如果文件集合不为空
    for (int i = 0; i < list.size(); i++) { //循环遍历集合
        java.io.File file = (java.io.File) list.get(i); //获取集合中文件对象
        //如果文件集中的路径名与用户选择的用户扩展名相同
        if (file.getAbsolutePath().toString().endsWith(
            jComboBox1.getSelectedItem().toString())) {
            String filePath = file.getAbsolutePath(); //获取文件完整路径
            //获取最后一次出现“\”索引的位置
            int index1 = filePath.lastIndexOf("\\");
            //获取最后一次出现“.”索引的位置
            int index2 = filePath.lastIndexOf(".");
            //获取文件名
            String fileName = filePath.substring(index1, index2);
            String backFile = filePath.substring(index2, filePath
                .length());
            fileheald.setEnd(filePath, fileSaveTextField.getText()
                + fileName + "change" + backFile, endingComboBox
                .getSelectedItem().toString()); //调用文件编码格式的转换方法
            bi = 2;
        }
    }
}
if (bi == 1) { //如果没有符合的文件, 给出提示信息
    JOptionPane.showMessageDialog(this, "没有符合条件的文件", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
    return;
}
JOptionPane.showMessageDialog(this, "文件编码格式转换完毕", "提示对话框",
    JOptionPane.WARNING_MESSAGE);
}
```

## 13.11 压缩和解压缩文件

### 13.11.1 功能概述

文件分割模块添加了文件压缩与解压缩功能。通过本模块的文件压缩与解压缩子模块可以将指定文件压缩成 ZIP 文件, 并可以对 ZIP 文件进行解压。



385

### 13.11.2 实现压缩文件



通过单击“工具”菜单中的“压缩文件夹”菜单项可实现对整个文件夹进行压缩。在



压缩文件夹窗体中可以指定文件压缩路径和压缩成功后的文件保存路径。“压缩文件模块”对话框如图 13.26 所示。

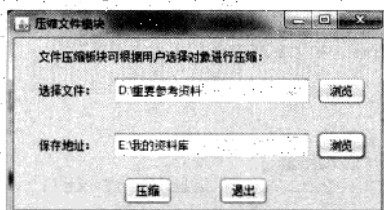


图 13.26 “压缩文件模块”对话框

完成文件的压缩主要应用了 ZIP 压缩技术,首先编写压缩文件的方法,具体代码如下:

```
//该方法以要进行压缩的文件路径及压缩后文件的保存路径作为参数
public void zip(String zipFileName,String inputFile){
    zip(zipFileName,new File(inputFile));    //调用 zip()方法
}
//该方法根据要进行压缩的文件路径,以压缩文件的保存路径创建 File 对象
public void zip(String zipFileName,File inputFile){
    ZipOutputStream out;
    try {
        //创建 ZipOutputStream 对象
        out = new ZipOutputStream(new FileOutputStream(zipFileName));
        zip(out,inputFile,"");    //调用 zip()方法
        out.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

由于对一些大的文件进行压缩会需要一些时间,为了避免程序出现死机现象,程序在单独的线程中实现了文件的压缩。当用户单击“压缩”文件夹后,首先判断用户是否输入合法的压缩文件地址和文件保存地址,并开启线程,具体代码如下:

```
private void zipinJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //判断压缩文件路径和文件保存路径是否为空

        if(fronzipTextField.getText().equals("")||saveTextField.getText().equals("")){
            int index = JOptionPane.showConfirmDialog(this, "请将信息添加完整", "消息对话框", JOptionPane.YES_NO_OPTION);
            if(index == JOptionPane.YES_OPTION){
                return;
            }
            if(index == JOptionPane.NO_OPTION){
                return;
            }
        }

        Thread thread = new Thread(this);
        thread.start();    //启动线程
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```







在线程的 run() 方法中调用压缩文件的方法，具体代码如下：

```
public void run() {
    try {
        MyZipMessage zipMessage = new MyZipMessage();
        zipMessage.setVisible(true);           //显示自定义提示信息窗体
        String strfrom = fronzipTextField.getText(); //获取压缩文件路径
        int index = strfrom.lastIndexOf("\\");
        //获取压缩文件的文件名
        String strnew = strfrom.substring(index, strfrom.length());
        try {
            testZip.zip(saveTextField.getText() + strnew + ".zip",
                fronzipTextField.getText());
            //调用压缩文件方法
            zipMessage.setVisible(false);
            JOptionPane.showMessageDialog(this, "文件压缩完毕提示", "消息对话框",
                JOptionPane.WARNING_MESSAGE);
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 13.11.3 实现文件解压

对于 ZIP 文件用户可以通过本模块中提供的解压功能进行文件解压，用户可以通过单击“工具”菜单中的“解压文件”进行解压缩，该模块运行结果如图 13.27 所示。

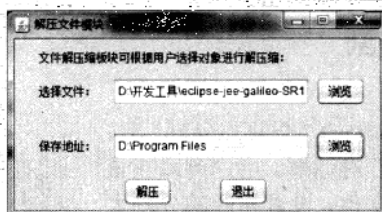


图 13.27 解压文件的运行结果

用户单击“浏览”按钮，可选择进行解压的文件夹。在文件选择器中，笔者设计了文件过滤器只允许用户选择 ZIP 文件进行解压。当用户单击“压缩”按钮后，会调用压缩文件方法，具体代码如下：

```
private void zipinJButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //判断用户是否将解压文件路径、文件的保存路径添加完成
        if(fronzipTextField.getText().equals("") || saveTextField.
            getText().equals("")){
            int index = JOptionPane.showConfirmDialog(this, "请将信息添加完整", "
            消息对话框", JOptionPane.YES_NO_OPTION);
            if(index == JOptionPane.YES_OPTION){
                return;
            }
            if(index == JOptionPane.NO_OPTION){
                return;
            }
        }
    }
}
```





```
        Thread thread = new Thread(this);           //开启线程方法
        thread.start();                             //启动线程
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void run() {
    MyOutZipMessage outZipMessage = new MyOutZipMessage();
                                //创建提示信息对话框
    outZipMessage.setVisible(true);
    com.cdd.uitl.TestZip testzip = new com.cdd.uitl.TestZip();
    //调用解压文件方法
    testzip.unzip(fronzipTextField.getText(), saveTextField.getText());
    outZipMessage.setVisible(false);
    JOptionPane.showMessageDialog(this, "文件解压完毕提示", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
}
```

## 13.12 文件分割与合并

### 13.12.1 功能概述

对于较大的文件在移动、存储等方面的操作都很不方便，如果将其分割成几个小文件，分别进行操作就可以解决问题了，本模块实现了对文件的分割与合成的功能。本节将介绍文件分割模块中实现文件分割与合并的过程。

### 13.12.2 实现文件分割

当用户单击“工具”菜单中的“合并与分割”后，程序将弹出“分割/合并模块”对话框。用户可以在该对话框中输入要进行分割的文件，设置相应的分割大小，选择路径后即可对文件进行分割，如图 13.28 所示。

通过文件分割子模块对任意格式的文件进行分割，分割后的文件，不能够单独运行，只有将分割后的小文件进行合并后，文件才可以继续运行，合并后的文件不影响原文件的性能。

首先编写文件分割方法，完成文件的分割是通过输入/输出流实现的。由于可以对任意格式的文件进行分割，应采用二进制流来完成文件的读取与写入。在文件分割方法中，需要指定要进行分割的文件，以及文件分割后保存的地址、分割后文件的大小（以 MB 为单位）。具体代码如下：

```
public void fenGe(File commFile, File untieFile, int filesize) {
    FileInputStream fis = null;
    int size=1024*1024;           //用来指定分割文件要以 MB 为单位
    try {
```

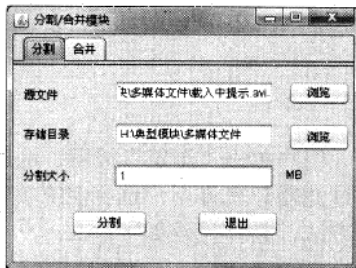


图 13.28 “分割/合并模块”对话框



388







```

        if (!untieFile.isDirectory()) { //如果要保存分割文件的路径不存在
            untieFile.mkdirs(); //创建该路径
        }
        size = size * filesize;
        int length = (int) commFile.length(); //获取文件大小
        int num = length / size; //获取文件大小除以 MB 的得数
        int yu = length % size; //获取文件大小与 MB 相除的余数
        //获取保存文件的完成路径信息
        String newfengeFile = commFile.getAbsolutePath();
        int fileNew = newfengeFile.lastIndexOf(".");
        String strNew = newfengeFile.substring(fileNew, newfengeFile
            .length()); //截取字符串
        fis = new FileInputStream(commFile); //创建 FileInputStream 类对象
        File[] fl = new File[num + 1]; //创建文件数组
        int begin = 0;
        for (int i = 0; i < num; i++) { //循环遍历数组
            fl[i] = new File(untieFile.getAbsolutePath() + "\\\" + (i + 1)
                + strNew + ".tem"); //指定分割后小文件的文件名
            if (!fl[i].isFile()) { //创建该文件
                fl[i].createNewFile();
            }
            FileOutputStream fos = new FileOutputStream(fl[i]);
            byte[] bl = new byte[size];
            fis.read(bl); //读取分割后的小文件
            fos.write(bl); //写文件
            begin = begin + size * 1024 * 1024;
            System.out.println("BEGIN " + begin);
            fos.close(); //关闭流
        }
        if (yu != 0) { //文件大小与指定文件分割大小相除的余数不为 0
            fl[num] = new File(untieFile.getAbsolutePath() + "\\\"
                //指定文件分割后数组中最后一个文件名
                + (num + 1) + strNew + ".tem");
            if (!fl[num].isFile()) {
                fl[num].createNewFile(); //新建文件
            }
            FileOutputStream fyu = new FileOutputStream(fl[num]);
            byte[] byt = new byte[yu];
            fis.read(byt);
            fyu.write(byt);
            fyu.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

在“分割/合并模块”对话框的“分割”选项卡中单击“分割”按钮，程序将调用文件分割方法，实现文件的分割。在文件调用文件分割方法时，需要注意的是，由于用户可能在“分割大小”文本框中输入非数字的数，因此笔者使用了异常处理机制来解决用户输入的非数字字符，具体代码如下：

```

private JButton getJButton2() { //实例化“分割”按钮方法
    if (commJButton == null) {
        commJButton = new JButton("分割");
        commJButton.setBounds(new Rectangle(60, 152, 80, 28));
        commJButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                int a = 0;
            }
        });
    }
}

```





```

//判断用户是否输入合法的文件名与分割后文件的保存地址
if ((fenGeFile == null) || (cunMuLu == null)) {
    JOptionPane.showMessageDialog(jFrame, "请将文件信息添加完整",
        "消息对话框", JOptionPane.WARNING_MESSAGE);
    return;
}
try {
    //获取用户选择文件的分割大小
    a = Integer.parseInt(commSize.getText());
    FengGeHeBing fgb = new FengGeHeBing();
    fgb.fenGe(fenGeFile, cunMuLu, a); //调用分割大小
    JOptionPane.showMessageDialog(jFrame, "文件分割成功",
        "消息对话框", JOptionPane.WARNING_MESSAGE);
} catch (NumberFormatException ee) {
    JOptionPane.showMessageDialog(jFrame,
        "请输入分割文件的大小为数字", "信息对话框",
        JOptionPane.WARNING_MESSAGE);
}
});
return commJButton;
}

```

### 13.12.3 实现文件合并

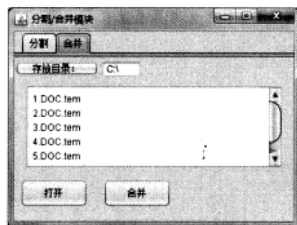


图 13.29 “合并”选项卡

通过文件合并工具，可将通过文件分割工具分割后的文件进行合并，运行效果如图 13.29 所示。

首先编写文件合并方法，实现文件合并仍然使用二进制流实现，具体代码如下：

```

/**
 * @param file: 要进行分割的文件数组对象
 * @param cunDir: 合并后文件的保存路径
 * @param hz: 合并后文件的格式
 */
public void heBing(File[] file, File cunDir, String hz) {
    try {
        File heBingFile = new File(cunDir.getAbsolutePath() + "\\UNTIE"
            + hz); //指定分割后文件的文件名
        if (!heBingFile.isFile()) {
            heBingFile.createNewFile();
        }
        //创建 FileOutputStream 对象
        FileOutputStream fos = new FileOutputStream(heBingFile);
        for (int i = 0; i < file.length; i++) { //循环遍历要进行合并的文件数组对象
            FileInputStream fis = new FileInputStream(file[i]);
            int len = (int) file[i].length(); //获取文件长度

```







```

        byte[] bRead = new byte[len];
        fis.read(bRead);           //读取文件
        fos.write(bRead);          //写入文件
        fis.close();               //将流关闭
    }
    fos.close();
    JOptionPane.showMessageDialog(jframe, "文件合并成功", "消息对话框",
        JOptionPane.WARNING_MESSAGE);
} catch (Exception e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(jframe, "您选择的文件不是分割文件, 请选择分割文件",
        "消息对话框", JOptionPane.WARNING_MESSAGE);
}
}
}

```

在“分割/合并模块”对话框的“合并”选项卡中, 单击“存放路径”按钮来确定合并后的文件的保存路径。当用户单击“打开”按钮后, 可以选择要进行合并的文件, 并将用户选择的文件名添加到如图 13.29 所示的文本域中。具体代码如下:

```

private JButton get getopenFilejButton () {           //实例化“打开”按钮方法
    if (openFilejButton == null) {
        openFilejButton = new JButton("打开");
        openFilejButton.setBounds(new Rectangle(8, 155, 85, 35));
        openFilejButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jFileChooser1.setSelectionMode(0); //设置选择器可以选择文件
                //运行选择器可以选择多个对象
                jFileChooser1.setMultiSelectionEnabled(true);
                //指定选择器为打开文件选择器
                int state=jFileChooser1.showOpenDialog(null);
                String s="";
                if(state==0){
                    //获取用户选择的文件对象集合
                    fl=jFileChooser1.getSelectedFiles();
                    int[] st=new int[fl.length];
                    for(int i=0;i<fl.length;i++){           //循环遍历文件集合
                        s=s+fl[i].getName()+"\r\n";
                        jTextArea.setText(s);
                    }
                    Arrays.sort(st);                         //将数组进行排序
                    //实例化文件合并数组对象, 为文件合并做准备
                    chuanFile=new File[st.length];
                    for(int i=0;i<st.length;i++){           //循环为数组复制
                        String strPath = fl[i].getAbsolutePath();
                        int fristPath = strPath.indexOf(".");
                        int lasePath = strPath.lastIndexOf(".");
                        String newPath = strPath.substring(fristPath, lasePath);
                        chuanFile[i]=new File(fl[i].getParent()+"\\"+(i+1)+
                            newPath+".tem");
                    }
                }
            }
        });
    }
    return openFilejButton;
}

```

用户确定了要进行合并的文件后, 单击“合并”按钮, 程序将调用文件合并方法, 实现对用户选择的合并文件进行合并, 具体代码如下:





```
private JButton getJButton5() { //实例化“合并”按钮方法
    if (jButton5 == null) {
        jButton5 = new JButton("合并");
        jButton5.setBounds(new Rectangle(113, 155, 85, 35));
        jButton5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                try{
                    FengGeHeBing fgb = new FengGeHeBing();//创建包装分割与合并类对象
                    String substr = null;
                    //循环遍历用户选择要进行合并的数组
                    for(int i = 0;i<chuanFile.length;i++){
                        String str = chuanFile[i].getAbsolutePath();
                        int strFrist = str.indexOf(".");
                        int strLase = str.lastIndexOf(".");
                        substr = str.substring(strFrist, strLase);
                    }
                    fgb.heBing(chuanFile,cunDir,substr); //调用合并文件的方法
                    jTextArea.setForeground(Color.red); //将文本域中的文字设置为红色
                }
                catch (Exception ee) {
                }
            }
        });
    }
    return jButton5;
}
```

## 13.13 实现文件分类管理

### 13.13.1 功能概述

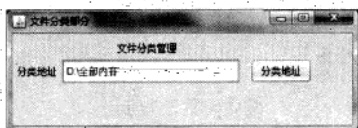


图 13.30 文件分类管理

本模块实现了文件分类整理的功能，当用户有大量的不同类型的文件要进行分类时，通过本模块进行分类十分方便、快捷。使用文件管理模块的分类整理功能，可以将选择的文件夹下的文件按照文件类型进行分类，并分别放到以文件扩展名为名称的文件夹中。文件分类管理界面运行结果如图 13.30 所示。

### 13.13.2 分类管理文件

实现文件的分类管理需要检索用户选择的文件夹中的文件，获取文件扩展名，并将相同文件格式的文件放置在以该文件扩展名命名的文件夹中。用户单击“分类地址”按钮，程序将弹出“文件选择”对话框；用户可以选择进行分类的地址。程序将在单独的线程中进行文件分类管理。“分类地址”按钮的单击事件，具体代码如下：

```
private void fileShowButtonActionPerformed(java.awt.event.ActionEvent evt) {
```





```

MyFileChooser myfile = new MyFileChooser();
myfile.getFolder();
String strPath = myfile.getFolderPath(); //获取“选择文件夹”对话框
pathTextField.setText(strPath); //获取文件选择路径
int n = JOptionPane.showConfirmDialog(this, "确定分类地址吗?", "确认对话框",
JOptionPane.YES_NO_OPTION); //设置“文件路径”文本框显示信息
if(n == JOptionPane.YES_OPTION){
statejLabel.setText("正在整理文件...");
Thread thread = new Thread(this); //启动线程
thread.start();
}
}

```

在线程的 `run()` 方法中, 实现文件的分类管理。首先获取用户选择路径中的文件扩展名, 并按照文件扩展名创建文件夹, 并将对应的文件复制到相应的文件夹中。具体代码如下:

```

public void run() {
FileHeald fileH = new FileHeald(); //创建封装文件操作类对象
//获取用户选择文件夹中的所有文件集合
List list = fileH.getList(pathTextField.getText());
for(int i = 0; i < list.size(); i++){ //循环遍历该文件集合
String strFile = list.get(i).toString();
int index = strFile.lastIndexOf(".");
if(index != -1){
//对文件夹进行截取, 获取文件扩展名
String strN = strFile.substring(index+1, strFile.length());
int ind = strFile.lastIndexOf("\\");
String strFileName = strFile.substring(ind, index);
//调用创建文件夹的方法, 新建文件夹
fileH.createFolder(pathTextField.getText()+"\\"+"分类");
fileH.createFolder(pathTextField.getText()+"\\"+"分类"+"\\"+strN);
if(strFile.endsWith(strN)){
//将文件集中与文件夹名称相同的文件复制到相应的文件夹中
fileH.copyFile(strFile, pathTextField.getText()+"\\"+"分类"+"\\"+strN+ "\\"+strFileName+strFile.substring(index, strFile.length()));
}
statejLabel.setText("文件分类结束");
}
}
}

```



# 第 14 章

---

## 图书管理模块

( Swing+SQL Server 2005 实现 )

进入 21 世纪以来，信息技术得到了飞速发展，人们的知识水平和技能也在不断提高，这就需要通过图书和网络资源来获取相应的知识和技能，在网络还没有完全普及的今天，图书就显得尤为重要，因此为了方便人们对图书的管理，可以开发一个图书管理模块，通过该模块可以快速查找到需要的图书，从而可以为用户提供方便、节省时间。通过阅读本章，读者可以学习到：

- » 添加图书类别
- » 修改和删除图书类别
- » 添加图书信息
- » 修改和删除图书信息
- » 查询图书信息





## 14.1 图书管理模块概述

### 14.1.1 模块概述

图书管理模块是一个用于企业和个人的图书管理程序,通过该程序可以实现添加图书分类,修改和删除图书分类,按照图书分类添加图书信息,修改图书信息及查询图书信息等操作,从而可以方便企业和个人对图书的充分利用,避免由于查找图书而浪费不必要的时间。

### 14.1.2 功能结构

图书管理程序可以实现添加图书类别,修改和删除图书类别,添加图书信息,修改和删除图书信息,以及查询图书信息等功能,其功能结构图如图 14.1 所示。

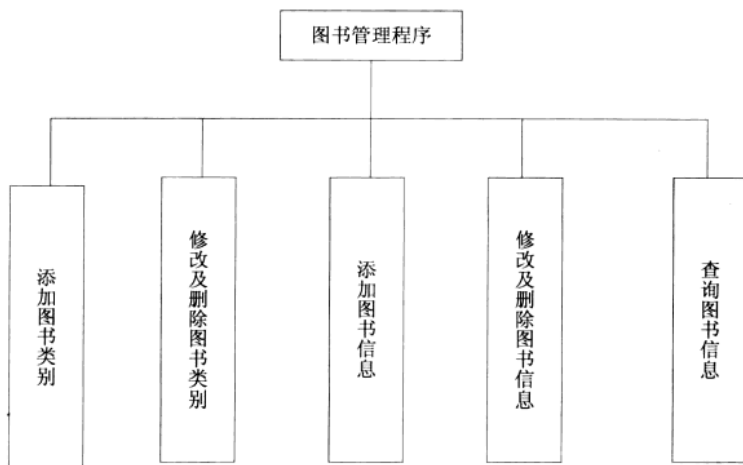


图 14.1 图书管理程序的功能结构图

### 14.1.3 程序预览

图书管理程序由主窗体,添加图书类别,修改和删除图书类别,添加图书信息,修改和删除图书信息,查询图书信息等部分组成,下面将对各组成部分的运行效果进行预览。

主窗体的运行效果如图 14.2 所示,主要用于通过菜单或工具栏打开图书管理模块的功能窗体,如打开添加图书类窗体、打开修改和删除图书类别窗体等。





图 14.2 “图书管理模块”主窗体运行效果

“添加图书类别”窗体的运行效果如图 14.3 所示，主要功能是用于在图书管理模块中添加图书类别。

“图书类别修改与删除”窗体的运行效果如图 14.4 所示，主要功能是用于修改和删除已经添加到图书管理模块中的图书类别。



图 14.3 “图书类别添加”窗体运行效果

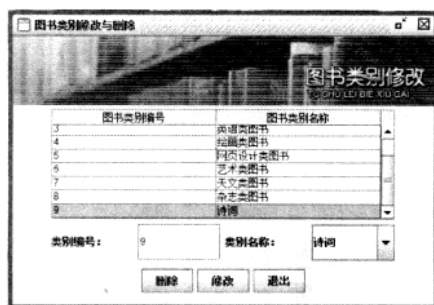


图 14.4 “图书类别修改与删除”窗体运行效果

“添加图书信息”窗体的运行效果如图 14.5 所示，主要功能是用于在图书管理模块中添加图书信息。



图 14.5 “图书信息添加”窗体运行效果

“图书信息修改与删除”窗体的运行效果如图 14.6 所示，主要功能是用于修改和删







除已经添加到图书管理模块中的图书信息。



图 14.6 “图书信息修改与删除”窗体运行效果

“图书查询”窗体的运行效果如图 14.7 和图 14.8 所示, 主要功能是用于查询图书信息和显示全部图书信息。

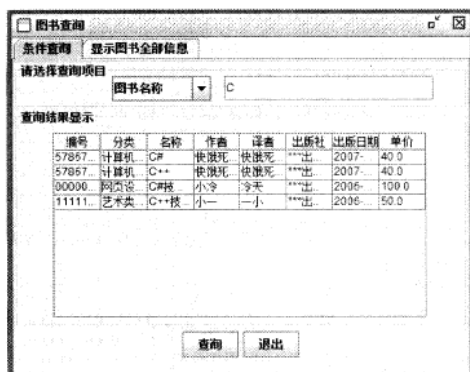


图 14.7 “条件查询”选项卡



图 14.8 “显示图书全部信息”选项卡

## 14.2 关键技术

### 14.2.1 连接和操作数据库

为方便用户对数据库进行操作, 可以创建一个进行数据库连接及操作的 Dao 类, 该类包括连接数据库的方法 getConnection()、执行查询语句的方法 executeQuery()、执行更新操作的方法 executeUpdate()、关闭数据库连接的方法 close()。下面将详细介绍如何编写图书管理模块中的数据库连接及操作的类 Dao.java。

(1) 指定类 Dao.java 保存的包, 并导入所需的类包, 本例将其保存到 com.zzk.dao 包中, 关键代码如下:





```

package com.zzk.dao;           //指定类的包名称
import java.sql.Connection;    //导入进行数据库连接时所使用的 java.sql.Connection 类
//导入进行数据库连接时所使用的 java.sql.DriverManager 类
import java.sql.DriverManager;
import java.sql.ResultSet;     //导入进行数据表查询时所使用的 java.sql.ResultSet 类
//导入进行数据库操作时捕捉异常使用的 java.sql.SQLException 类
import java.sql.SQLException;

```



包语句以关键字 `package` 后面紧跟一个包名称, 然后以分号 “;” 结束; 包

语句必须出现在 `import` 语句之前; 一个 `.java` 文件只能有一个包语句。

(2) 在 `Dao.java` 类的构造方法中创建数据库连接操作。在此类中首先定义数据库连接驱动包名、数据库连接路径、数据库连接用户名、密码等静态变量, 然后在构造函数中实现数据库连接操作。在数据库连接代码中需要添加 `try...catch` 关键字, 捕捉数据库连接时可能抛出的异常。关键代码如下:

```

//定义驱动包名称
protected static String dbClassName = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
protected static String dbUrl = "jdbc:sqlserver://localhost:1433;"
    + "DatabaseName=db_library;SelectMethod=Cursor"; //定义数据库连接路径
protected static String dbUser = "sa";                //定义数据库连接用户名
protected static String dbPwd = "";                   //定义数据库连接密码

protected static String second = null;
private static Connection conn = null;                //定义一个数据库连接

private Dao() {
    try {
        //捕捉数据库连接异常
        //如果连接为空
        if (conn == null) {
            Class.forName(dbClassName).newInstance(); //装载 SQL Server 驱动
            //获取数据库连接
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPwd);
        }
        else
            //如果连接不为空
            //返回
            return;
    } catch (Exception ee) {
        ee.printStackTrace(); //捕捉数据库连接异常
    }
}

```

(3) 创建执行查询语句的方法 `executeQuery`, 其返回值为 `ResultSet` 结果集。首先需要初始化 `Dao` 对象, 调用构造函数, 从而获取数据库连接。有一点值得注意, 就是在创建数据库连接之前首先判断数据库连接是否为空, 如果为空再创建数据库连接, 避免造成程序资源的浪费。`executeQuery` 方法的代码如下:

```

private static ResultSet executeQuery(String sql) {
    try {
        //捕捉数据库操作异常
        //数据库连接如果为空
        if (conn == null)
            new Dao();
        return conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            //返回一个 ResultSet 结果集
            ResultSet.CONCUR_UPDATABLE).executeQuery(sql);
    } catch (SQLException e) {
        e.printStackTrace(); //捕捉异常
        return null;
    } finally {

```







```
}
}
```

(4) 创建执行更新操作的方法 `executeUpdate()`，它的返回值为 `int` 型的整数，此返回值代表数据表更新操作是否成功，返回 1 代表成功，返回 -1 代表没有成功。`executeUpdate()` 方法的关键代码如下：

```
private static int executeUpdate(String sql) {
    try {
        if(conn==null)                //捕捉数据库操作异常
            new Dao();                //如果数据库连接为空
        return conn.createStatement().executeUpdate(sql); //获取数据库连接
    } catch (SQLException e) {         //进行数据库更新操作
        System.out.println(e.getMessage()); //打印捕捉的异常
        return -1;                     //返回-1
    } finally {
    }
}
```

(5) 为了避免运行程序时资源的浪费，优化项目运行速度，需要在完成数据库操作后，关闭数据库连接，所以，笔者在 `Dao.java` 类中创建了关闭数据库连接的方法 `close()`。为了使数据库连接在程序结束后确定会被关闭，在 `close()` 方法中加入了 `finally` 字段，在 `finally` 块中将数据库连接置空。关键代码如下：

```
public static void close() {
    try {
        conn.close();                //捕捉异常
    } catch (SQLException e) {        //关闭数据库连接
        e.printStackTrace();         //捕捉异常
    } finally {
        conn = null;                //在最终执行块中将数据库连接置空
    }
}
```

## 14.2.2 MenuActions 类的编写

通常，激活同一个命令有多种方式，用户可以通过工具栏中按钮、菜单选择特定的功能。在本系统中，最常用的命令就是弹出内部窗体，笔者将本系统中需要弹出的内部窗体命令统一放入 `MenuActions` 类中，这样触发任何一种组件事件时，都会按照统一的方式处理。

Swing 包提供了一个非常有用的机制，用来封装命令，并将其连接到多个事件源，这种机制就是 `Action` 接口。`Action` 接口有如下方法：

```
public void actionPerformed(ActionEvent e)
public Object getValue(String key)
public void putValue(String key, Object value)
public boolean isEnabled()
public void setEnabled(boolean b)
public void addPropertyChangeListener(PropertyChangeListener listener)
public void removePropertyChangeListener(PropertyChangeListener listener)
```

其中，第一个方法在实现 `ActionListener` 接口的程序中经常看到，实际上 `Action` 接口扩展了 `ActionListener` 接口。



399





getValue()与 putValue()方法用来存储与提取动作对象的预定义名称与值，例如：

```
//将图标存储到动作对象中
action.putValue(Action.SMALL_ICON,new ImageIcon("*.gif"));
```

表 14.1 列举了几种常用的动作对象的预定义名称。

表 14.1 动作对象的预定义名称

名 称	值
NAME	名称，显示在按钮或菜单上
SMALL_ICON	小图标，显示在按钮或菜单上
SHORT_DESCRIPTION	简单提示说明，当鼠标放在按钮或菜单上出现提示
LONG_DESCRIPTION	详细提示说明

setEnabled()方法用于开启或禁用动作对象，isEnabled()方法用于检查动作是否启用。

实现 Action 接口需要将接口中的所有方法都实现，所以在通常情况下都使用实现该接口的 AbstractAction 类，本系统中的 MenuActions 类正是继承了 AbstractAction 类，在 MenuActions 类中只要重写 AbstractAction 类中的 actionPerformed()方法即可。MenuAction 类的完整代码如下：

```
package com.zzk;

import java.awt.event.ActionEvent;
import java.util.HashMap;
import java.util.Map;
import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.JFrame;
import com.zzk.iframe.BookAddIFrame;
import com.zzk.iframe.BookModiAndDelIFrame;
import com.zzk.iframe.BookSearchIFrame;
import com.zzk.iframe.BookTypeAddIFrame;
import com.zzk.iframe.BookTypeModiAndDelIFrame;
import com.zzk.util.*;

/**
 * 菜单和按钮的 Action 对象
 */
public class MenuActions {
    private static Map<String, JFrame> frames; // 子窗体集合

    public static BookSearchAction BOOK_SEARCH; // 图书搜索窗体动作
    public static BookTypeModiAction BOOKTYPE_MODIFY; // 图书类型修改窗体动作
    public static BookTypeAddAction BOOKTYPE_ADD; // 图书类型添加窗体动作
    public static BookModiAction BOOK_MODIFY; // 图书信息修改窗体动作
    public static BookAddAction BOOK_ADD; // 图书信息添加窗体动作
    public static ExitAction EXIT; // 模块退出动作

    static {
        frames = new HashMap<String, JFrame>();
        BOOK_SEARCH = new BookSearchAction();
        BOOKTYPE_MODIFY = new BookTypeModiAction();
        BOOKTYPE_ADD = new BookTypeAddAction();
        BOOK_MODIFY = new BookModiAction();
    }
}
```





```
BOOK_ADD = new BookAddAction();
EXIT = new ExitAction();
}

private static class BookSearchAction extends AbstractAction {
    BookSearchAction() {
        super("图书搜索", null);
        putValue(Action.LONG_DESCRIPTION, "搜索入库的图书信息");
        putValue(Action.SHORT_DESCRIPTION, "图书搜索");
    }

    public void actionPerformed(ActionEvent e) {
        if (!frames.containsKey("图书查询") || frames.get("图书查询").
            isClosed()) {
            BookSearchIFrame iframe = new BookSearchIFrame();
            frames.put("图书查询", iframe);
            Library.addIFame(frames.get("图书查询"));
        }
    }
}

private static class BookTypeModiAction extends AbstractAction {
    BookTypeModiAction() {
        super("图书类别修改", null);
        putValue(Action.LONG_DESCRIPTION, "修改图书的类别信息");
        putValue(Action.SHORT_DESCRIPTION, "图书类别修改");
    }

    public void actionPerformed(ActionEvent e) {
        if (!frames.containsKey("图书类别修改")
            || frames.get("图书类别修改").isClosed()) {
            BookTypeModiAndDelIFrame iframe = new BookTypeModiAndDelIFrame();
            frames.put("图书类别修改", iframe);
            Library.addIFame(frames.get("图书类别修改"));
        }
    }
}

private static class BookTypeAddAction extends AbstractAction {
    BookTypeAddAction() {
        super("图书类别添加", null);
        putValue(Action.LONG_DESCRIPTION, "添加新的图书类别");
        putValue(Action.SHORT_DESCRIPTION, "图书类别添加");
    }

    public void actionPerformed(ActionEvent e) {
        if (!frames.containsKey("图书类别添加")
            || frames.get("图书类别添加").isClosed()) {
            BookTypeAddIFrame iframe = new BookTypeAddIFrame();
            frames.put("图书类别添加", iframe);
            Library.addIFame(frames.get("图书类别添加"));
        }
    }
}

// 图书修改与删除
private static class BookModiAction extends AbstractAction {
    BookModiAction() {
        super("图书信息修改", null);
        putValue(Action.LONG_DESCRIPTION, "修改和删除图书信息");
    }
}
```





```
        putValue(Action.SHORT_DESCRIPTION, "图书信息修改");
    }

    public void actionPerformed(ActionEvent e) {
        if (!frames.containsKey("图书信息修改")) {
            || frames.get("图书信息修改").isClosed()) {
                BookModiAndDelIFrame iframe = new BookModiAndDelIFrame();
                frames.put("图书信息修改", iframe);
                Library.addIFame(frames.get("图书信息修改"));
            }
        }
    }

    private static class BookAddAction extends AbstractAction { // 图书信息添加
        BookAddAction() {

            super("图书信息添加", null);
            // super();
            putValue(Action.LONG_DESCRIPTION, "添加新的图书信息");
            putValue(Action.SHORT_DESCRIPTION, "图书信息添加");
        }

        public void actionPerformed(ActionEvent e) {
            if (!frames.containsKey("图书信息添加")) {
                || frames.get("图书信息添加").isClosed()) {
                    BookAddIFrame iframe = new BookAddIFrame();
                    frames.put("图书信息添加", iframe);
                    Library.addIFame(frames.get("图书信息添加"));
                }
            }
        }
    }

    private static class ExitAction extends AbstractAction { // 退出
        public ExitAction() {
            super("退出模块", null);
            putValue(Action.LONG_DESCRIPTION, "退出图书管理模块");
            putValue(Action.SHORT_DESCRIPTION, "退出模块");
        }

        public void actionPerformed(final ActionEvent e) {
            System.exit(0);
        }
    }

    private MenuActions() {
        super();
    }
}
```



下面以“图书信息修改”菜单项为例对 MenuActions 类的功能进行说明，以下代码是在 MenuActions 类中创建的一个内部类，这个内部类用于创建菜单栏中的“图书信息修改”菜单项的动作对象，在此类的构造函数中创建组件的提示说明，在 actionPerformed()方法执行时弹出“图书信息修改与删除”窗体。关键代码如下：

```
// 图书修改与删除
private static class BookModiAction extends AbstractAction {
    BookModiAction() {
        super("图书信息修改", null);
    }
}
```





```

        putValue(Action.LONG_DESCRIPTION, "修改和删除图书信息");
        putValue(Action.SHORT_DESCRIPTION, "图书信息修改");
    }

    public void actionPerformed(ActionEvent e) {
        if (!frames.containsKey("图书信息修改")) {
            || frames.get("图书信息修改").isClosed() {
                BookModiAndDelIFrame iframe = new BookModiAndDelIFrame();
                frames.put("图书信息修改", iframe);
                Library.addIFame(frames.get("图书信息修改"));
            }
        }
    }
}

```

将此内部类的对象作为 MenuActions 类的成员变量, 然后再使用 static 定义一个静态区域进行初始化。类在被加载时, 首先执行 static 定义的静态区域内部的代码, 且只会被执行一次。关键代码如下:

```

public static BookModiAction BOOK_MODIFY;    // 图书信息修改窗体动作
static {
    BOOK_MODIFY = new BookModiAction();      // 初始化图书信息修改内部类的对象
}

```

同理, 菜单栏中其他菜单项与子菜单中的菜单项也是以相同方式被封装到 MenuActions 类中的。当某个组件需要使用这个动作对象时, 以按钮为例, 可以使用如下代码实现:

```

JButton bookModiAndDelButton=new JButton(MenuActions.BOOK_MODIFY);

```

### 14.2.3 限制文本框长度类的编写

在 Swing 语言创建的窗体中, 当 JTextField 组件创建时, 可以指定文本框的宽度, 例如:

```

JPanel panel=new JPanel();                //创建面板
JTextField textField=new JTextField(20);    //创建文本框
panel.add(textField);                      //将文本框添加到面板中

```

在 JTextField 的构造器中设定的宽度并不是用户能输入的字符个数上限, 用户可以在文本框中输入一个更长的字符串, 此时需要限制用户输入字符串的长度, 笔者创建了限制文本框输入长度的类 MyDocument.java。创建此类的步骤如下。

(1) 创建 MyDocument.java 类, 此类继承了 PlainDocument 类, 关键代码如下:

```

public class MyDocument extends PlainDocument{
}

```

(2) 在 MyDocument.java 类中创建两个构造函数, 其中一个是有参数的, 另一个是无参数的, 关键代码如下:

```

public MyDocument(int newMaxLength){        //设置文本框的最大长度
    super();                                //执行父类构造方法
    maxLength = newMaxLength;               //将参数赋予类成员变量
}

public MyDocument(){                        //无参的构造函数
    this(10);                              //将数值 10 赋予类成员变量
}

```





(3) 重载父类方法 `insertString()`，在此方法中限定文本框允许输入字符串长度，关键代码如下：

```
public void insertString(int offset, String str, AttributeSet a)
    throws BadLocationException {

    if (getLength() + str.length() > maxLength) {    //这里假定限制长度为10
        return;                                       //返回
    } else {
        super.insertString(offset, str, a);
    }
}
```

(4) 在程序设计中，当需要限制用户输入字符串长度时，可以使用如下代码：

```
JTextField textField = new JTextField("请输入13位书号",13); //初始化文本框
//设置“书号”文本框最大输入值为13位
textField.setDocument(new MyDocument(13));
```

#### 14.2.4 描述组合框索引与内容类的编写

在程序编写的过程中，经常会遇到组合框组件的应用。有时要在窗体中的组合框中显示具体内容，通常需要在数据库中存储此组合框的索引值，这时便需要使用一种数据结构将组合框中的内容与索引值联系在一起。`java.util.Map` 形式是比较好的选择，可以使用 `Map` 接口中的 `put()` 方法将索引值与具体内容放入集合中，当得到索引值时获取具体内容可以使用 `Map` 接口中的 `get(key)` 方法。描述组合框索引与内容类的编写步骤如下。

(1) 创建组合框组件的索引值与其所对应的内容的 `Item.java` 类，这个类中不仅包含代表组合框索引的成员变量 `id` 和代表组合框内容的成员变量 `name`，还包括这两个成员变量的 `setXXX()` 和 `getXXX()` 方法。关键代码如下：

```
package com.zzk.JComPz;
public class Item {
    public String id;           //组合框索引值
    public String name;        //组合框内容

    public String getId() {     //id对应的 getXXX() 方法
        return id;
    }

    public void setId(String id) { // id对应的 setXXX() 方法
        this.id = id;
    }

    public String getName() {   //name对应的 getXXX() 方法
        return name;
    }

    public void setName(String name) { //name对应的 setXXX() 方法
        this.name = name;
    }

    public String toString() {  //重写 Object 类中的 toString() 方法
        return getName();
    }
}
```







(2) 创建 MapPz.java 类, 使用 Map 关联组合框的索引与组合框的具体内容。这里以图书类别编号与图书类别创建组合框为例, 首先在此类中初始化 Map 集合, 取图书类别相关内容, 将图书类别相关内容放入 Item 类中; 然后使用 put() 方法将图书类别编号与图书类别名称分别放入 Map 集合中; 最后返回类型为 Map 的集合。关键代码如下:

```
package com.zzk.JComPz;
public class MapPz {
    static Map map = new HashMap();           //初始化 Map 接口

    public static Map getMap() {
        List list = Dao.selectBookCategory(); //获取图书类别相关内容
        for (int i = 0; i < list.size(); i++) { //循环操作
            BookType booktype = (BookType) list.get(i); //取得集合中的值
            Item item = new Item(); //初始化 Item 对象
            item.setId(booktype.getId()); //将图书类别编号放入 Item 类中
            item.setName(booktype.getTypeName()); //将图书类别名称放入 Item 类中
            //将图书类别编号与 item 对象放入 map
            map.put(item.getId(), item);
        }
        return map; //返回集合
    }
}
```

(3) 上述代码中用到了 Dao.java 中的 selectBookCategory() 方法, 此方法用于查询图书类别相关信息, 首先将数据库查询的相关信息放入 JavaBean 中, 然后将 JavaBean 对象添加到 list 集合中, 最终将结果以 List 形式返回。关键代码如下:

```
public static List selectBookCategory() {
    List list = new ArrayList(); //初始化 List 对象
    String sql = "select * from tb_bookType"; //查询图书类别表 SQL 语句
    //执行 SQL 语句, 返回 ResultSet 对象
    ResultSet rs = Dao.executeQuery(sql);
    try {
        while (rs.next()) { //循环结果集
            BookType bookType = new BookType(); //初始化 BookType 对象
            //将数据库中查询的 id 值赋予到 JavaBean 中
            bookType.setId(rs.getString("id"));
            //将数据库中查询 typeName 值赋予到 JavaBean 中
            bookType.setTypeName(rs.getString("typeName"));
            //将数据库中查询的 days 值赋予到 JavaBean 中
            bookType.setDays(rs.getString("days"));
            //将数据库中查询的 fk 值赋予到 JavaBean 中
            bookType.setFk(rs.getString("fk"));
            list.add(bookType); //将 JavaBean 对象添加到 list 中
        }
    } catch (Exception e) {
        e.printStackTrace(); //捕捉异常
    }
    Dao.close(); //关闭数据库连接
    return list; //将集合返回
}
```





## 14.2.5 在 JLabel 上添加图片类

为了美化窗体，通常需要在窗体上添加图片，一般情况下使用如下方式添加图片：  
在窗体上添加 JPanel。

- (1) 在 JPanel 上添加 JLabel。
- (2) 将图片初始化为 ImageIcon 对象。
- (3) 使用 JLabel.setIcon(ImageIcon) 代码实现在窗体上添加图片功能。

在这里将上述操作封装在公共类中，命名为 CreatedIcon.java 类，在此类中定义一个返回 ImageIcon 类对象的方法，此方法以当前图片的文件名称为参数初始化一个 ImageIcon 类对象，关键代码如下：

```
package com.zzk.util;
public class CreatedIcon {
    public static ImageIcon add(String ImageName){    //返回 ImageIcon 类型的对象
        //当前图片的路径
        URL IconUrl = Library.class.getResource("/"+ImageName);
        ImageIcon icon=new ImageIcon(IconUrl);    //将路径封装到 ImageIcon 对象中
        return icon;    //返回 icon 对象
    }
}
```



Library.class.getResource("/1.jpg") 指代的图片为项目名称下的 res 文件下的图片，实际上 "/" 指代的路径为项目名称中的 res 文件。

当需要在 JLabel 中添加图片时，可以使用如下代码：

```
final JLabel headLogo = new JLabel();    //创建 JLabel 对象
//使用 CreatedIcon 类中的 add() 方法返回一个 ImageIcon 对象
ImageIcon bookModiAndDelIcon=CreatedIcon.add("bookModiAndDel.jpg");
headLogo.setIcon(bookModiAndDelIcon);    //设置 JLabel 图片
```

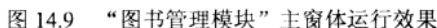
## 14.3 主窗体

### 14.3.1 功能概述

图书管理程序的主窗体是一个 JFrame 窗体，用户可以通过菜单和工具栏打开具体的功能窗体，每个功能窗体都是一个 JInternalFrame 窗体，也就是功能窗体只能显示在主窗体内部，而且会随着主窗体的移动而移动，当主窗体最小化时功能窗体也随之最小化，主窗体的运行效果如图 14.9 所示。







在图书管理程序的主窗体中可以通过菜单打开功能窗体，进行相应的图书管理操作，为此需要创建菜单栏，并为菜单栏添加菜单和菜单项，主窗体类 `Library` 中用于创建菜单栏的 `createMenu()` 方法的代码如下：

407



### 14.3.3 实现工具栏的设计

在图书管理程序的主窗体除了可以通过菜单打开功能窗体外,还可以通过工具栏上的工具按钮打开功能窗体,进行相应的图书管理操作,为此需要创建工具栏,并为工具栏添加工具按钮,主窗体类 Library 中用于创建工作栏的 createToolBar()方法的代码如下:

```
/**
 * 创建工具栏
 * @return JToolBar
 */
private JToolBar createToolBar() { // 创建工具栏的方法
    JToolBar toolBar = new JToolBar();
    toolBar.setFloatable(false);
    toolBar.setBorder(new BevelBorder(BevelBorder.RAISED));
    JButton bookAddButton=new JButton(MenuActions.BOOK_ADD);
    //创建图标
    ImageIcon icon=new ImageIcon(Library.class.getResource("/bookAddtb.jpg"));
    bookAddButton.setIcon(icon);
    bookAddButton.setHideActionText(true);
    toolBar.add(bookAddButton);
    //在工具栏中添加图书修改与删除图标
    JButton bookModiAndDelButton=new JButton(MenuActions.BOOK_MODIFY);
    ImageIcon bookmodiicon=CreatecdIcon.add("bookModiAndDeltb.jpg"); //创建图标
    bookModiAndDelButton.setIcon(bookmodiicon);
    bookModiAndDelButton.setHideActionText(true);
    toolBar.add(bookModiAndDelButton);
    JButton bookTypeAddButton=new JButton(MenuActions.BOOKTYPE_ADD);
    ImageIcon bookTypeAddicon=CreatecdIcon.add("bookTypeAddtb.jpg"); //创建图标
    bookTypeAddButton.setIcon(bookTypeAddicon);
    bookTypeAddButton.setHideActionText(true);
    toolBar.add(bookTypeAddButton);
    JButton ExitButton=new JButton(MenuActions.EXIT);
    ImageIcon Exiticon=CreatecdIcon.add("exittb.jpg"); //创建图标
    ExitButton.setIcon(Exiticon);
    ExitButton.setHideActionText(true);
    toolBar.add(ExitButton);
    return toolBar;
}
```

### 14.3.4 为窗体添加背景

为了使图书管理程序的主窗体美观,可以为主窗体添加背景,这可以通过为桌面面板类 JDesktopPane 的实例添加带有图片的 JLabel 标签实现,下面是为主窗体添加背景的现代码:

```
// Library 类的静态成员桌面面板
private static final JDesktopPane DESKTOP_PANE = new JDesktopPane();
public Library() { // Library 类的构造方法
    super();
    // ...省略了部分代码
    final JLabel label = new JLabel();
    label.setBounds(0, 0, 0, 0);
    label.setIcon(null); // 窗体背景
}
```







```

DESKTOP_PANE.addComponentListener(new ComponentAdapter() {
    public void componentResized(final ComponentEvent e) {
        Dimension size = e.getComponent().getSize();
        label.setSize(e.getComponent().getSize());
        label.setText("<html></html>");
    }
});
DESKTOP_PANE.add(label, new Integer(Integer.MIN_VALUE));
getContentPane().add(DESKTOP_PANE);
}

```

## 14.4 添加图书类别

### 14.4.1 功能概述

在图书管理程序中为了方便对图书进行分类管理,可以在添加图书信息时为图书指定类别,而手动输入的效率低,并且在用户打字速度很慢的情况下,还会影响图书信息的录入速度,为此可以将图书类别先保存起来,然后通过下拉列表框进行选择,从而可以提高录入速度,因而需要创建“图书类别添加”窗体,“图书类别添加”窗体的运行效果如图 14.10 所示,在该窗体中完成图书类别的添加。



图 14.10 “图书类别添加”窗体运行效果

### 14.4.2 保存图书类别

在图书管理程序中创建用于保存图书类别的内部窗体类 `BookTypeAddIframe`, 然后为该窗体添加控件, 其中, “保存”按钮的动作事件用于保存图书类别信息, 其事件代码如下:

```

button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        if (bookTypeName.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "图书类别文本框不可为空");
            return;
        }
        int i = Dao.InsertBookType(bookTypeName.getText().trim());
    }
});

```



409





```
        if(i==1){
            JOptionPane.showMessageDialog(null, "添加成功!");
            doDefaultCloseAction();
        }
    }
});
```

上述“保存”按钮的动作事件中调用了 Dao 类的 insertBookType 方法，该方法用于将保存图书类别信息的 SQL 语句传递给 Dao 类的 executeUpdate 方法，insertBookType 方法的代码如下：

```
/**
 * 保存图书类别的方法
 * @param bookTypeName 图书类别名称
 * @return
 */
public static int InsertBookType(String bookTypeName) {
    int i = 0;
    try {
        String sql = "insert into tb_bookType(typeName) values('"
            + bookTypeName + "')";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return i;
}
```

上述代码调用了 Dao 类的 executeUpdate 方法，该方法用于执行参数指定的 SQL 语句，上面代码执行的是将图书类别保存到数据库的 SQL 语句，该方法的代码如下：

```
private static int executeUpdate(String sql) {
    try {
        if (conn == null)
            new Dao();
        return conn.createStatement().executeUpdate(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return -1;
    } finally {
    }
}
```

## 14.5 修改和删除图书类别

### 14.5.1 功能概述



当用户单击主窗体中的“基础数据维护”\“图书类别管理”\“图书类别修改”菜单项，可以打开图书类别修改与删除窗体，如图 14.11 所示，该窗体的主要功能是用于修改和删除已经添加到图书管理模块中的图书类别，方法是从表格中选择需要修改或删除的图书类别信息，然后通过表格下面的文本框和下拉列表框可以对图书类别进行修改，并通过“修改”按钮保存所做的修改；如果需要删除图书类别，只需要从表格中选择需要删除的





图书类别信息，然后单击“删除”按钮，即可完成图书类别的删除。

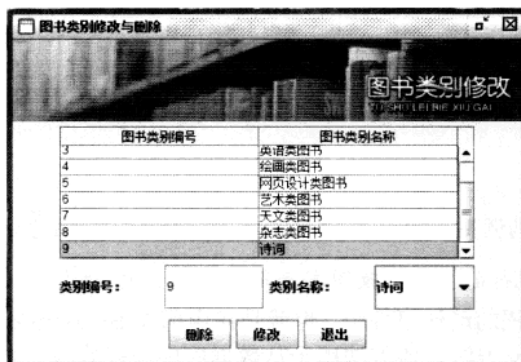


图 14.11 “图书类别修改与删除”窗体运行效果

### 14.5.2 修改图书类别

在图书管理程序中创建用于修改和删除图书类别的内部窗体类 `BookTypeModiAndDellFrame`，然后为该窗体添加控件，其中，“修改”按钮的动作事件用于保存对图书类别信息所做的修改，其实现代码是通过内部类 `ButtonAddListener` 实现的，该内部类的完整代码如下：

```
class ButtonAddListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
        Object selectedItem = bookTypeModel.getSelectedItemAt();
        int i=Dao.UpdatebookType(BookTypeId.getText().trim(),selectedItem.
        toString());
        if(i==1){
            JOptionPane.showMessageDialog(null, "修改成功");
            Object[][] results=getFileStates(Dao.selectBookCategory());
            model.setDataVector(results,columnNames);
            table.setModel(model);
        }
    }
}
```

上述“修改”按钮的动作事件中，也就是在内部类 `ButtonAddListener` 中调用了 `Dao` 类的 `UpdatebookType` 方法，该方法用于将修改图书类别信息的 SQL 语句传递给 `Dao` 类的 `executeUpdate` 方法，`UpdatebookType` 方法的代码如下：

```
/**
 * 用于修改图书类别的方法
 * @param id 图书类别 ID
 * @param typeName 图书类别名称
 * @return
 */
public static int UpdatebookType(String id, String typeName) {
    int i = 0;
    try {
        String sql = "update tb_bookType set typeName='" + typeName
```





```
        + "' where id='" + id + "'";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return i;
}
```



上述代码调用了 Dao 类的 executeUpdate 方法,该方法用于执行参数指定的 SQL 语句,上面代码执行的就是修改图书类别信息的 SQL 语句。

另外在“修改”按钮的动作事件中,也就是在内部类 ButtonAddListener 中调用了 Dao 类的 selectBookCategory 方法,该方法用于从数据库中获得修改后的图书类别信息,使窗体的表格中显示的信息与所做的修改同步,该方法的代码如下:

```
/**
 * 获得图书类别信息的方法
 * @return
 */
public static List selectBookCategory() {
    List list = new ArrayList();
    String sql = "select * from tb_bookType";
    ResultSet rs = Dao.executeQuery(sql);
    try {
        while (rs.next()) {
            BookType bookType = new BookType();
            bookType.setId(String.valueOf(rs.getInt("id")));
            bookType.setTypeNames(rs.getString("typeNames"));
            list.add(bookType);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return list;
}
```

上述代码中使用了 BookType 类的实例来存储图书类别信息,并将该实例存储在 List 集合中,BookType 类的完整代码如下:

```
package com.zzk.model;
public class BookType {
    private String id; // 类别编号
    private String typeNames; // 类别名称
    // 下面是成员变量的 getter 方法和 setter 方法
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getTypeNames() {
        return typeNames;
    }
    public void setTypeNames(String typeNames) {
        this.typeNames = typeNames;
    }
}
```







```
}
}
```

### 14.5.3 删除图书类别

修改和删除图书类别内部窗体中“删除”按钮的动作事件，用于从表格中删除当前选择的图书类别信息，并更新数据库表中的数据记录，其事件代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        int i=Dao.DelbookType(BookTypeId.getText().trim());
        if(i==1){
            JOptionPane.showMessageDialog(null, "删除成功");
            Object[][] results=getFileStates(Dao.selectBookCategory());
            model.setDataVector(results,columnNames);
            table.setModel(model);
        }
    }
});
```



在“删除”按钮的动作事件中也调用了 Dao 类的 selectBookCategory 方法，该方法用于从数据库中获得删除图书类别后的信息，使窗体的表格中显示的信息与所做的删除操作同步。

在上述“删除”按钮的动作事件中还调用了 Dao 类的 DelbookType 方法，该方法用于根据图书类别 ID 删除图书类别信息，该方法的代码如下：

```
/**
 * 删除图书类别信息的方法
 * @param id 图书类别 ID
 * @return
 */
public static int DelbookType(String id) {
    int i = 0;
    try {
        String sql = "delete from tb_bookType where id='" + id + "'";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return i;
}
```



上述代码调用了 Dao 类的 executeUpdate 方法，该方法用于执行参数指定的 SQL 语句，上面代码执行的就是根据图书类别 ID 删除图书类别信息的 SQL 语句。





## 14.6 添加图书信息

### 14.6.1 功能概述

在图书管理程序中通过“图书信息添加窗体”来添加图书信息，运行效果如图 14.12 所示，当用户输入完图书信息后，单击“添加”按钮完成图书信息的添加操作。



图 14.12 “图书信息添加”窗体运行效果

### 14.6.2 保存图书信息

在图书管理程序中创建用于保存图书信息的内部窗体类 `BookAddIframe`，然后为该窗体添加控件，并为“添加”按钮创建动作事件监听器类 `addBookActionListener`，该类是窗体类 `BookAddIframe` 的内部类，用于响应“添加”按钮的动作事件，完成图书信息的添加操作。类 `addBookActionListener` 的完整代码如下：

```
// “添加”按钮的单击事件监听器
class addBookActionListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        if (ISBN.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "书号文本框不可以为空");
            return;
        }
        if (ISBN.getText().length() != 13) {
            JOptionPane.showMessageDialog(null, "书号文本框输入位数为 13 位");
            return;
        }
        if (bookName.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "图书名称文本框不可以为空");
            return;
        }
        if (writer.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "作者文本框不可以为空");
            return;
        }
        if (pubDate.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "出版日期文本框不可以为空");
            return;
        }
        if (price.getText().length() == 0) {
```







```

        JOptionPane.showMessageDialog(null, "单价文本框不可以为空");
        return;
    }
    String ISBNs=ISBN.getText().trim();
    //分类
    Object selectedItem = bookType.getSelectedItemAt();
    if (selectedItem == null)
        return;
    Item item = (Item) selectedItem;
    String bookTypes=item.getId();
    String translators=translator.getText().trim();
    String bookNames=bookName.getText().trim();
    String writers=writer.getText().trim();
    String publishers=(String)publisher.getSelectedItemAt();
    String pubDates=pubDate.getText().trim();
    String prices=price.getText().trim();
    int i=Dao.Insertbook(ISBNs,bookTypes, bookNames, writers, translators,
        publishers, java.sql.Date.valueOf(pubDates),Double.parseDouble
        (prices));
    if(i==1){
        JOptionPane.showMessageDialog(null, "添加成功");
        doDefaultCloseAction();
    }
}
}
}

```

在上述“添加”按钮的动作事件监听器类 `addBookActionListener` 中,调用了 `Dao` 类的 `insertBook` 方法,该方法用于保存所添加的图书信息, `insertBook` 方法的代码如下:

```

/*
 * 保存图书信息方法
 */
public static int Insertbook(String ISBN, String typeId, String bookname,
    String writer, String translator, String publisher, Date date,
    Double price) {

    int i = 0;
    try {
        String sql = "insert into tb_bookInfo(ISBN,typeId,bookname,writer,translator,
            publisher,date,price) values("
            + ISBN
            + ","
            + typeId
            + ","
            + bookname
            + ","
            + writer
            + ","
            + translator
            + ","
            + publisher
            + ","
            + date
            + "," + price + ")";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    Dao.close();
    return i;
}

```





上述代码调用了 Dao 类的 executeUpdate 方法,该方法用于执行参数指定的 SQL 语句,上面代码执行的是将图书信息保存到数据库的 SQL 语句。

## 14.7 修改和删除图书信息

### 14.7.1 功能概述

当用户单击主窗体中的“基础数据维护”\“图书信息管理”\“图书信息修改”菜单项,可以打开“图书信息修改与删除”窗体,如图 14.13 所示,该窗体的主要功能是由于修改和删除已经添加到图书管理模块中的图书信息,方法是从表格中选择需要修改或删除的图书信息,然后通过表格下面的文本框和下拉列表框等控件对图书信息进行修改,并通过“修改”按钮保存所做的修改;如果需要删除图书信息,只需要从表格中选择需要删除的图书信息,然后单击“删除”按钮,即可完成图书信息的删除。



图 14.13 “图书信息修改与删除”窗体

### 14.7.2 修改图书信息

在图书管理程序中创建用于修改和删除图书信息的内部窗体类 BookModiAndDelIIFrame,然后为该窗体添加控件,其中,“修改”按钮的动作事件用于保存对图书信息所做的修改,其实现代码是通过内部类 addBookActionListener 实现的,该内部类的完整代码如下:

```
class addBookActionListener implements ActionListener {
    public void actionPerformed(final ActionEvent e) {
        // 修改图书信息表
        if (ISBN.getText().length() == 0) {
            JOptionPane.showMessageDialog(null, "书号文本框不可以为空或输入数字不可以大于 13 个");
        }
    }
}
```





```

        return;
    }
    if (ISBN.getText().length() != 13) {
        JOptionPane.showMessageDialog(null, "书号文本框输入位数为 13 位");
        return;
    }
    if (bookName.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "图书名称文本框不可以为空");
        return;
    }
    if (writer.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "作者文本框不可以为空");
        return;
    }
    if (publisher.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "出版人文本框不可以为空");
        return;
    }
    //日期与单价进行数字验证代码?
    if (pubDate.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "出版日期文本框不可以为空");
        return;
    }
    if (price.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "单价文本框不可以为空");
        return;
    }
    String ISBNs = ISBN.getText().trim();
    //分类
    Object selectedItem = bookTypeModel.getSelectedItemAt();
    if (selectedItem == null)
        return;
    Item item = (Item) selectedItem;
    String bookTypes = item.getId();
    System.out.println(bookTypes);
    String translators = translator.getText().trim();
    String bookNames = bookName.getText().trim();
    String writers = writer.getText().trim();
    String publishers = publisher.getText().trim();
    String pubDates = pubDate.getText().trim();
    String prices = price.getText().trim();
    int i = Dao.Updatebook(ISBNs, bookTypes, bookNames, writers, translators,
        publishers, Date.valueOf(pubDates), Double.parseDouble(prices));
    if (i == 1) {
        JOptionPane.showMessageDialog(null, "修改成功");
        Object[][] results = getFileStates(Dao.selectBookInfo());
        // 使用表格模型
        DefaultTableModel model = new DefaultTableModel();
        table.setModel(model);
        model.setDataVector(results, columnNames);
    }
}
}

```

上述“修改”按钮的动作事件中，也就是在内部类 `addBookActionListener` 中调用了 `Dao` 类的 `Updatebook` 方法，该方法用于将修改图书信息的 SQL 语句传递给 `Dao` 类的 `executeUpdate` 方法，`Updatebook` 方法的代码如下：

```

/*
 * 修改图书信息方法

```





```
*/
public static int Updatebook(String ISBN, String typeId, String bookname,
    String writer, String translator, String publisher, Date date,
    Double price) {
    int i = 0;
    try {
        String sql = "update tb_bookInfo set ISBN='" + ISBN + "',typeId='"
            + typeId + "',bookname='" + bookname + "',writer='"
            + writer + "',translator='" + translator + "',publisher='"
            + publisher + "',date='" + date + "',price=" + price
            + " where ISBN='" + ISBN + "'";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return i;
}
```



上述代码调用了 Dao 类的 executeUpdate 方法,该方法用于执行参数指定的 SQL 语句,上面代码执行的就是修改图书信息的 SQL 语句。

另外,在“修改”按钮的动作事件中,也就是在内部类 addBookActionListener 中调用了 Dao 类的 selectBookInfo 方法,该方法用于从数据库中获得修改后的图书信息,使窗体的表格中显示的信息与所做的修改同步,该方法的代码如下:

```
/*
 * 查询图书信息
 */
public static List selectBookInfo() {
    List list = new ArrayList();
    String sql = "select * from tb_bookInfo";
    ResultSet rs = Dao.executeQuery(sql);
    try {
        while (rs.next()) {
            BookInfo bookinfo = new BookInfo();
            bookinfo.setISBN(rs.getString("ISBN"));
            bookinfo.setTypeid(rs.getString("typeid"));
            bookinfo.setBookname(rs.getString("bookname"));
            bookinfo.setWriter(rs.getString("writer"));
            bookinfo.setTranslator(rs.getString("translator"));
            bookinfo.setPublisher(rs.getString("publisher"));
            bookinfo.setDate(rs.getDate("date"));
            bookinfo.setPrice(rs.getDouble("price"));
            list.add(bookinfo);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return list;
}
```







### 14.7.3 删除图书信息

修改和删除图书内部窗体中“删除”按钮的动作事件,用于从表格中删除当前选择的图书信息,并更新数据库表中的数据记录,其事件代码如下:

```
button_2.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        String ISBNs=ISBN.getText().trim();
        int i=Dao.Delbook(ISBNs);
        if(i==1){
            JOptionPane.showMessageDialog(null, "删除成功");
            Object[][] results=getFileStates(Dao.selectBookInfo());
            // 使用表格模型
            DefaultTableModel model=new DefaultTableModel();
            table.setModel(model);
            model.setDataVector(results, columnNames);
        }
    }
});
```



在“删除”按钮的动作事件中也调用了 Dao 类的 selectBookInfo 方法,该方法用于从数据库中获得删除图书后的信息,使窗体的表格中显示的信息与所做的删除操作同步。

在上述“删除”按钮的动作事件中还调用了 Dao 类的 Delbook 方法,该方法用于根据图书的 ISBN 删除图书信息,该方法的代码如下:

```
/**
 * 删除图书信息的方法
 * @param ISBN 图书的 ISBN
 * @return
 */
public static int Delbook(String ISBN) {
    int i = 0;
    try {
        String sql = "delete from tb_bookInfo where ISBN='" + ISBN + "'";
        i = Dao.executeUpdate(sql);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return i;
}
```



上述代码调用了 Dao 类的 executeUpdate 方法,该方法用于执行参数指定的 SQL 语句,上面代码执行的就是根据图书 ISBN 删除图书信息的 SQL 语句。





## 14.8 查询图书信息

### 14.8.1 功能概述

当用户单击主窗体中的“查询管理”\“图书搜索”菜单项,可以打开“图书查询”窗体,如图 14.14 所示,用户指定查询条件后,单击“查询”按钮可以将满足条件的图书信息查询出来,并在窗体上的表格中显示所查询的信息;单击窗体上的“显示图书全部信息”选项卡,可以显示图书的全部信息,效果如图 14.15 所示。



图 14.14 “条件查询”选项卡



图 14.15 “显示图书全部信息”选项卡

### 14.8.2 查询满足条件的图书

在图书管理程序中创建用于查询图书信息的内部窗体类 `BookSearchIframe`, 然后为该窗体添加选项卡控件, 并在“条件查询”选项卡中完成查询满足条件的图书信息, 也就是在“查询”按钮的动作事件中实现根据用户指定的条件查询图书信息, “查询”按钮的事件代码如下:

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        String name=(String)choice.getSelectedItemAt();
        if(name.equals("图书名称")){
            Object[][] results=getselect(Dao.selectbookmohu(textField_1.
                getText()));
            table_2 = new JTable(results,booksearch);
            scrollPane_1.setViewportViewView(table_2);
        }
        else if(name.equals("图书作者")){
            Object[][] results=getselect(Dao.selectbookmohuwriter(textField_1.
                getText()));
            table_2 = new JTable(results,booksearch);
            scrollPane_1.setViewportViewView(table_2);
        }
    }
});
```







在“查询”按钮的动作事件中,可以根据图书名称和图书作者进行查询,如果需要根据图书名称进行查询,可以调用 Dao 类的 selectbookmohu 方法根据图书名称进行查询,如果需要根据图书作者进行查询,可以调用 Dao 类的 selectbookmohuwriter 方法根据图书作者进行查询。根据图书名称进行查询的 selectbookmohu 方法如下:

```
public static List selectbookmohu(String bookname) {
    List list = new ArrayList();
    String sql = "select * from tb_bookInfo where bookname like '%"
        + bookname + "%'";
    System.out.print(sql);
    ResultSet s = Dao.executeQuery(sql);
    try {
        while (s.next()) {
            BookInfo bookinfo = new BookInfo();
            bookinfo.setISBN(s.getString(1));
            bookinfo.setTypeid(s.getString(2));
            bookinfo.setBookname(s.getString(3));
            bookinfo.setWriter(s.getString(4));
            bookinfo.setTranslator(s.getString(5));
            bookinfo.setPublisher(s.getString(6));
            bookinfo.setDate(s.getDate(7));
            bookinfo.setPrice(s.getDouble(8));
            list.add(bookinfo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return list;
}
```

根据图书作者进行查询的 selectbookmohuwriter 方法如下:

```
public static List selectbookmohuwriter(String writer) {
    List list = new ArrayList();
    String sql = "select * from tb_bookInfo where writer like '%" + writer
        + "%'";
    System.out.print(sql);
    ResultSet s = Dao.executeQuery(sql);
    try {
        while (s.next()) {
            BookInfo bookinfo = new BookInfo();
            bookinfo.setISBN(s.getString(1));
            bookinfo.setTypeid(s.getString(2));
            bookinfo.setBookname(s.getString(3));
            bookinfo.setWriter(s.getString(4));
            bookinfo.setTranslator(s.getString(5));
            bookinfo.setPublisher(s.getString(6));
            bookinfo.setDate(s.getDate(7));
            bookinfo.setPrice(s.getDouble(8));
            list.add(bookinfo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return list;
}
```

在上述两个方法中使用了 Dao 类的 executeQuery 方法,该方法用于执行指定的 SQL 查询语句,并返回查询结果集,该方法的完整代码如下:

```
/**
 * 执行 SQL 查询语句,获得结果集的方法
```





```
* @param SQL 需要执行的 SQL 查询语句
* @return
*/
private static ResultSet executeQuery(String sql) {
    try {
        if (conn == null)
            new Dao();
        return conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE).executeQuery(sql);
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    } finally {
    }
}
```

### 14.8.3 显示全部图书信息

在图书管理程序的查询图书信息内部窗体中，为了能够在“显示图书全部信息”选项卡中显示全部图书的信息，可以在内部窗体类 BookSearchIFrame 的构造方法中添加如下代码：

```
final JPanel panel_2 = new JPanel();
tabbedPane.addTab("显示图书全部信息", null, panel_2, null);
scrollPane = new JScrollPane();
scrollPane.setPreferredSize(new Dimension(450, 250));
panel_2.add(scrollPane);
Object[][] results = getselect(Dao.selectbookserch()); // 获得表格中的数据集合
String[] booksearch = { "编号", "分类", "名称", "作者", "译者", "出版社", "出版日期", "单价" };
table_1 = new JTable(results, booksearch);
scrollPane.setViewportView(table_1);
```

上述代码中使用了 Dao 类的 selectbookserch 方法，该方法用于查询全部图书信息，其完整代码如下：

```
public static List selectbookserch() {
    List list = new ArrayList(); // 存储图书全部信息的 List 集合对象
    String sql = "select * from tb_bookInfo"; // 查询全部图书信息的 SQL 语句
    ResultSet s = Dao.executeQuery(sql); // 执行查询语句，获得结果集
    try {
        while (s.next()) { // 循环遍历结果集
            BookInfo bookinfo = new BookInfo(); // 创建 BookInfo 类的实例
            // 使用 BookInfo 类的实例存储每一条记录
            bookinfo.setISBN(s.getString(1));
            bookinfo.setTypeid(s.getString(2));
            bookinfo.setBookname(s.getString(3));
            bookinfo.setWriter(s.getString(4));
            bookinfo.setTranslator(s.getString(5));
            bookinfo.setPublisher(s.getString(6));
            bookinfo.setDate(s.getDate(7));
            bookinfo.setPrice(s.getDouble(8));
            list.add(bookinfo); // 将 BookInfo 类的实例存储到 List 集体对象中
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    Dao.close();
    return list;
}
```







上述代码中使用了 BookInfo 类的实例来存储图书信息，并将该实例存储在 List 集合中，BookInfo 类的完整代码如下：

```
package com.zzk.model;
import java.sql.Date;
public class BookInfo {
    private String ISBN;           // 图书的 ISBN
    private String typeid;         // 图书类型
    private String writer;         // 作者
    private String translator;     // 译者
    private String publisher;      // 出版社
    private Date date;             // 出版日期
    private Double price;          // 单价
    private String bookname;       // 书名
    // 下面是成员变量的 getter 方法和 setter 方法
    public String getBookname() {
        return bookname;
    }
    public void setBookname(String bookname) {
        this.bookname = bookname;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
    public String getISBN() {
        return ISBN;
    }
    public void setISBN(String isbn) {
        ISBN = isbn;
    }
    public Double getPrice() {
        return price;
    }
    public void setPrice(Double price) {
        this.price = price;
    }
    public String getPublisher() {
        return publisher;
    }
    public void setPublisher(String publisher) {
        this.publisher = publisher;
    }
    public String getTranslator() {
        return translator;
    }
    public void setTranslator(String translator) {
        this.translator = translator;
    }
    public String getTypeid() {
        return typeid;
    }
    public void setTypeid(String typeid) {
        this.typeid = typeid;
    }
    public String getWriter() {
        return writer;
    }
    public void setWriter(String writer) {
        this.writer = writer;
    }
}
```



# 第 15 章

---

## 五子棋游戏模块

( Swing+Socket 网络技术实现 )

五子棋是起源于中国古代的传统黑白棋，它不仅能增强人的思维能力，提高智力，而且还富含哲理，有助于修身养性。五子棋既有现代休闲的明显特征“短、平、快”，又有古典哲学的高深学问“阴阳易理”；既具有简单易学的特性，为人们所喜爱，又有深奥的技巧和高水平的国际性比赛。五子棋文化源远流长，具有东方的神秘和西方的直观；既有“场”的概念，亦有“点”的连接。五子棋起源于中国古代，发展于日本，风靡于欧洲，可以说五子棋是中西方文化的交流点，是古今哲学的结晶。在本章中，笔者将设计一个五子棋游戏模块，通过本章的学习，读者能够学到：

- » 绘制半透明的登录界面
- » 游戏记录回放
- » 绘制可以调整大小的五子棋棋盘
- » 网络连接状态检测





## 15.1 五子棋模块概述

### 15.1.1 模块概述

相信很多人都会下五子棋游戏,当游戏的一方构成 5 个连续的棋子,无论是水平方向、垂直方向,还是斜对角线方向,都表示获胜了。对于初学网络的开发人员来说,设计一个网络五子棋游戏再合适不过了。从规模上看,网络五子棋只需要包含客户端和服务端两个窗口,规模比较小,而本模块设计的是独立运行的客户端,不需要服务器。从功能上看,网络五子棋涉及两台主机间的通信,相互需要传递棋子信息、控制指令和文本信息,这需要定义一个应用协议来解释数据报,涉及网络开发的许多知识。

### 15.1.2 功能结构

运行五子棋游戏模块后,将显示登录窗体,用户需要输入昵称和对方 IP 地址才能进行游戏。游戏的主要功能包括下棋、悔棋、和棋及游戏回放,其功能结构如图 15.1 所示。

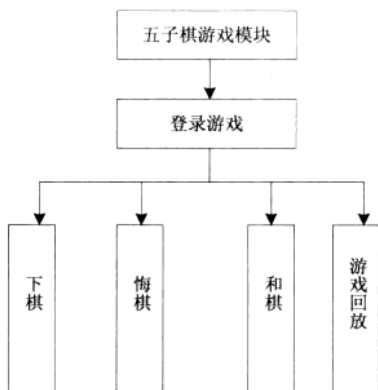


图 15.1 五子棋游戏模块功能结构图

### 15.1.3 系统预览

程序运行以后,首先显示登录界面,这个登录界面使用半透明效果将主窗体遮罩,然后显示登录界面,用户必须输入自己的昵称和对方主机的 IP 地址才能登录。程序运行效果如图 15.2 所示。





图 15.2 登录联机的程序界面

与对方建立网络连接之后，会进入游戏主窗体。单击“开始”按钮将开始进行游戏。当自己头像下方有一盒棋子时，就轮到自己下棋，棋子的颜色和自己头像下方棋盒里的棋子颜色相同。游戏主窗体的运行效果如图 15.3 所示。

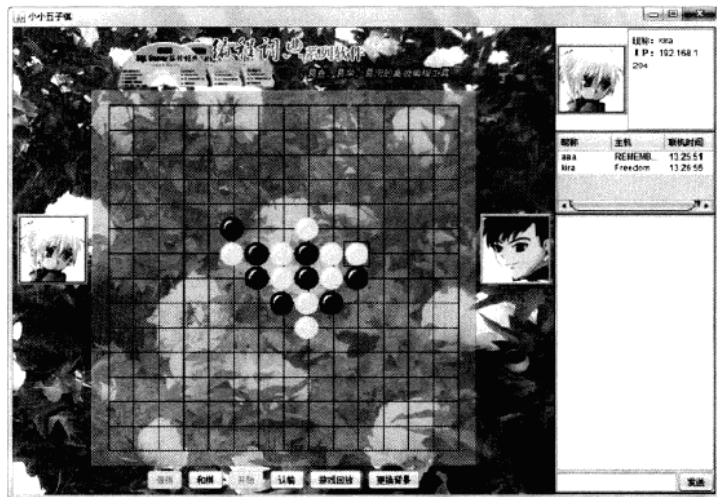


图 15.3 游戏主窗体运行效果

当游戏的一方胜利时，程序界面将提示“对方胜利”，并且把对方的 5 个连线棋子用星号标注，并禁止棋盘的落子行为。程序界面如图 15.4 所示。





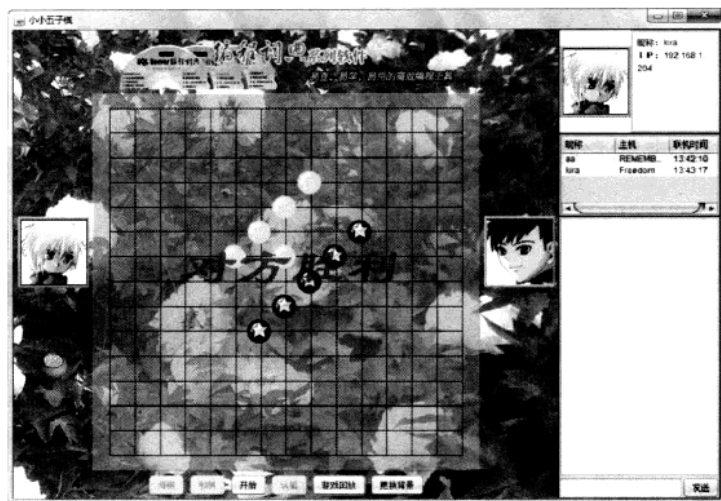


图 15.4 对方胜利后的效果

当自己一方有 5 个棋子连成一线时, 将提示“你胜利了”的信息, 并且将自己一方相连的 5 个棋子用星号标注, 效果如图 15.5 所示。

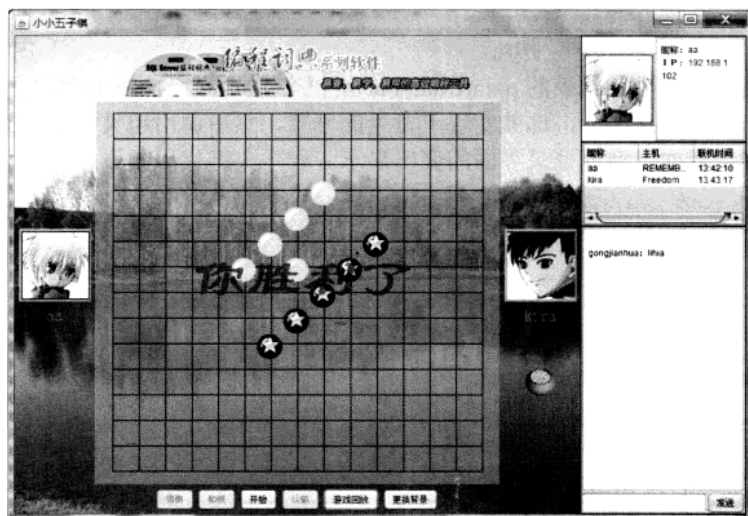


图 15.5 己方胜利后的效果

游戏的开始、悔棋、和棋、游戏回放等动作, 由棋盘下方的控制面板组成, 该面板还包含一个“更换背景”按钮可以更换程序界面的背景图片。控制面板如图 15.6 所示。



图 15.6 控制按钮面板





## 15.2 关键技术

### 15.2.1 实现透明的登录界面

本模块的登录面板实现了透明的效果,并且将登录面板下的主窗体用半透明遮罩效果变暗,而登录界面正常显示,这样,用户的注意力就会放在登录界面上。登录面板的关键效果如图 15.7 所示。

实现登录界面的关键技术,使用了 GlassPane 面板,它位于窗体的最顶层,Swing 默认该面板为隐藏模式。本模块继承 JPanel 类编写了登录面板,其中包含登录信息的文本框和“登录”按钮等信息,然后调用 JFrame 窗体的 setGlassPane()方法将该面板设置为 GlassPanel 玻璃面板。程序关键代码如下:

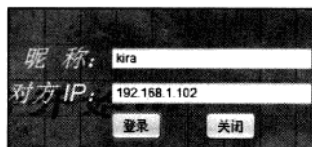


图 15.7 登录界面的背景半透明效果

```
public MainFrame() {  
    initComponents();  
    setGlassPane(loginPanel1);  
    loginPanel1.setVisible(true);  
}
```

// 初始化窗体界面  
// 设置登录面板为玻璃面板  
// 显示登录面板

接下来需要重写登录面板的 paintComponent()方法绘制登录面板的界面,使用 80%透明的矩形填充整个登录面板,这就实现了背景的遮罩,然后调用超类的 paintComponent()方法,绘制登录界面,实现登录界面的突出显示。程序关键代码如下:

```
protected void paintComponent(Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;  
    Composite composite = g2.getComposite();  
    // 设置绘图使用透明合成规则  
    g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,  
0.8f));  
    g2.fillRect(0, 0, getWidth(), getHeight());  
    g2.setComposite(composite);  
    super.paintComponent(g2);  
}
```

// 获取 2D 绘图上下文  
// 备份合成模式  
// 使用当前颜色填充矩形空间  
// 恢复原有合成模式  
// 执行超类的组件绘制方法

### 15.2.2 监控网络连接状态

在进行游戏的过程中,为了防止由于网络故障或某一方掉线使得游戏无法结束、无法重新开始,在网络五子棋模块添加了网络状态测试功能。实现网络状态测试功能比较简单,在建立网络连接后,调用 Socket 的 setOOBInline()方法启用紧急数据接收,然后 Socket 的另一端会调用 sendUrgentData()方法向自己发送紧急数据,同时本地也会执行 Socket 类的 sendUrgentData()方法向对方发送紧急数据,如果对方网络有故障或者掉线了,那么该方法会抛出异常,说明网络连接断开。程序运行结果如图 15.8 所示。





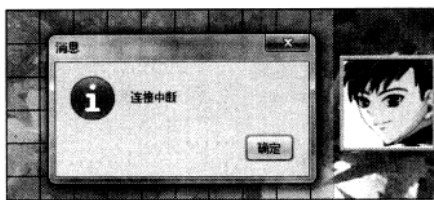


图 15.8 网络中断的提示界面

实现网络状态检测的程序关键代码如下:

```
try {
    // 省略部分代码
    serverSocket.setOOBInline(true); // 启用紧急数据的接收
    InputStream is = serverSocket.getInputStream(); // 获取网络输入流
    ObjectInputStream objis = new ObjectInputStream(is); // 创建对象输入流
    while (frame.isVisible()) {
        serverSocket.sendUrgentData(255); // 发送紧急数据
        // 省略部分代码
    }
} catch (SocketException ex) {
    Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    JOptionPane.showMessageDialog(frame, "连接中断"); // 提示用户网络中断
    frame.getChessPanell().reInit();
    DefaultTableModel model = (DefaultTableModel) frame.userInfoTable.getModel();
    model.setRowCount(0);
    frame.getGlassPane().setVisible(true);
} catch (IOException ex) {
    Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
} catch (ClassNotFoundException ex) {
    Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
}
```

### 15.2.3 绑定属性的 JavaBean

本模块定义了棋盘模型类 GobangModel, 它是一个 JavaBean, 用于记录棋盘上当前棋子的布局, 它使用一个二维数组保存所有棋子。

这个记录棋盘棋子数据的 JavaBean 将棋子数组定义为绑定属性, 当该属性被修改时会自动产生属性变更事件, 并通知所有监听该属性的监听器。在棋盘类中就定义了一个监视该属性的监听器, 它在棋盘模型的数据发生改变时立刻更新棋盘界面。

要实现 JavaBean 的绑定属性, 必须实现以下两个机制。

#### 1. 产生 PropertyChange 事件

无论任何情况, 只要 JavaBean 中的绑定属性发生了变化, 该 JavaBean 就必须发送一个 PropertyChange 属性改变事件给所有已经注册的事件监听器, 棋盘模型 JavaBean 在设置棋盘数组属性的 setChessmanArray()方法中产生了该事件。程序关键代码如下:

```
private PropertyChangeSupport propertySupport; // 定义属性工具类
private static GobangModel model; // 定义自身的变量
private byte[][] chessmanArray = new byte[15][15]; // 定义棋子数组
// 定义属性名称
public static final String PROP_CHESSMANARRAY = "chessmanArray";
```





```
public void setChessmanArray(byte[][] chessmanArray) {
    this.chessmanArray = chessmanArray;
    // 报告所有已注册监听器的绑定属性更新
    propertySupport.firePropertyChange(PROP_CHESSMANARRAY, null, chessmanArray);
}
```

## 2. 实现事件监听器的注册与注销

对棋盘数据模型 JavaBean 感兴趣的监听器必须通过该 JavaBean 提供的方法进行注册或注销，这两个方法分别是 `addPropertyChangeListener()` 方法和 `removePropertyChangeListener()` 方法。只有注册到该 JavaBean 的事件监听器才能监听 JavaBean 属性的改变事件。棋盘数据模型对这两个方法的实现代码如下：

```
public void addPropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.addPropertyChangeListener(listener); // 添加事件监听器
}
public void removePropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.removePropertyChangeListener(listener); // 移除事件监听器
}
```

## 15.2.4 在棋盘中绘制棋子

在设计网络五子棋时，需要在棋盘中绘制棋子，并且在窗口更新时保证棋子仍然在棋盘上。笔者采用的方式是定义一个二维数组，数组的大小与棋盘中表格的行和列相对应，描述棋盘中可以放置棋子的所有点。绘制棋子的界面效果如图 15.9 所示。

程序关键代码如下：

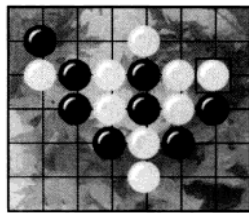


图 15.9 绘制棋子界面效果

```
byte[][] chessmanArray = gobangModel1.getChessmanArrayCopy();
for (int i = 0; i < chessmanArray.length; i++) { // 遍历棋盘数据模型绘制棋子
    for (int j = 0; j < chessmanArray[i].length; j++) {
        byte chessman = chessmanArray[i][j];
        int x = i * chessWidth;
        int y = j * chessHeight;
        if (chessman != 0)
            System.out.println("chess is:" + chessman);
        if (chessman == WHITE_CHESSMAN) { // 绘制白棋
            g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
        } else if (chessman == BLACK_CHESSMAN) { // 绘制黑棋
            g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
        } else if (chessman == (WHITE_CHESSMAN ^ 3)) { // 绘制最近的白棋落子
            g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
            g.drawRect(x, y, chessWidth, chessHeight);
        } else if (chessman == (BLACK_CHESSMAN ^ 3)) { // 绘制最近的黑棋落子
            g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
            g.drawRect(x, y, chessWidth, chessHeight);
        } // 绘制导致胜利的连线白棋
    } else if (chessman == ((byte) (WHITE_CHESSMAN ^ 8))) {
        g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
        g.drawImage(rightTop_img, x, y, chessWidth, chessHeight, this);
    }
}
```







```
// 绘制导致胜利的连线黑棋
} else if (chessman == (BLACK_CHESSMAN ^ 8)) {
    g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
    g.drawImage(rightTop_img, x, y, chessWidth, chessHeight, this);
}
}
```

### 15.2.5 实现动态调整棋盘大小

在设计网络五子棋时,为了突出游戏的特点,允许用户在游戏进行的过程中调整窗口的大小。效果如图 15.10 和图 15.11 所示。

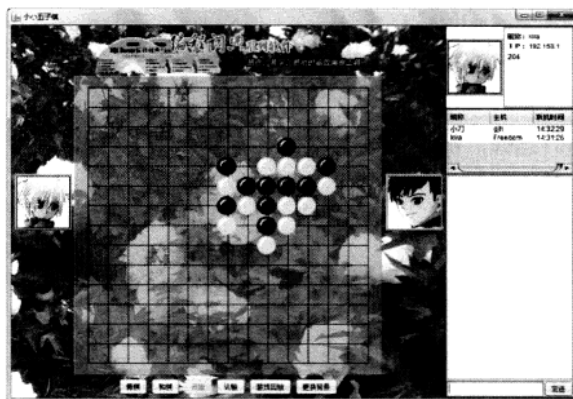


图 15.10 下棋窗口



图 15.11 棋盘缩放

实现该功能的难点在于窗口调整大小后,棋盘和棋子的大小需要调整,棋盘表格的大





小需要调整, 棋盘中当前棋子的位置需要调整。笔者采用的方式是根据棋盘面板的宽度和高度计算棋格和棋子大小, 然后使用计算出的棋格宽度和高度绘制棋盘, 使用计算出的棋子高度与宽度绘制指定大小的棋子。绘制棋盘的关键程序代码如下:

```
private void drawPanel(Graphics2D g) {
    Composite composite = g.getComposite();           // 备份合成规则
    Color color = g.getColor();                       // 备份前景颜色
    g.setComposite(AlphaComposite.SrcOver.derive(0.6f)); // 设置透明合成
    g.setColor(new Color(0xAABBAA));                 // 设置前景白色
    g.fillRect(0, 0, getWidth(), getHeight(), true); // 绘制半透明的矩形
    g.setComposite(composite);                       // 恢复合成规则
    g.setColor(color);                               // 恢复原来的前景色
    int w = getWidth();                             // 棋盘宽度
    int h = getHeight();                             // 棋盘高度
    int chessW = w / 15, chessH = h / 15;           // 棋子宽度和高度
    int left = chessW / 2 + (w % 15) / 2;           // 棋盘左边界
    int right = left + chessW * 14;                  // 棋盘右边界
    int top = chessH / 2 + (h % 15) / 2;            // 棋盘上边界
    int bottom = top + chessH * 14;                  // 棋盘下边界
    for (int i = 0; i < 15; i++) {
        // 画每条横线
        g.drawLine(left, top + (i * chessH), right, top + (i * chessH));
    }
    for (int i = 0; i < 15; i++) {
        // 画每条竖线
        g.drawLine(left + (i * chessW), top, left + (i * chessW), bottom);
    }
}
```

### 15.2.6 游戏悔棋

为了增加网络五子棋的灵活性, 在本模块中设计了悔棋功能。当用户想要悔棋时, 需要向对方发送悔棋请求, 如果对方同意悔棋, 则双方都进行悔棋操作, 如图 15.12 所示。

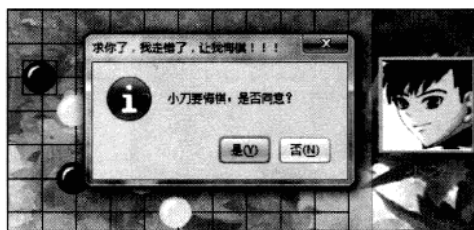


图 15.12 悔棋时确认界面



432



五子棋游戏模块设计了一个存储下棋步骤的双向队列, 要实现悔棋功能, 需要从该队列中弹出两步落子动作, 其中, 包括自己的下棋步骤和对家的下棋步骤。

```
public synchronized void repentOperation() {
    Deque<byte[][]> chessQueue = gobangPanel1.getChessQueue(); // 获取下棋队列
    if (chessQueue.isEmpty()) {
        return;
    }
    // 获取上两次走棋的棋谱
    for (int i = 0; i < 2 && !chessQueue.isEmpty(); i++) {
```





```

        chessQueue.pop(); // 废弃走棋步骤
    }
    if (chessQueue.size() < 1) {
        chessQueue.push(new byte[15][15]);
    }
    byte[][] pop = chessQueue.peek();
    GobangModel.getInstance().updateChessmanArray(pop); // 更新棋盘的棋子布局
    repaint();
}

```

### 15.2.7 游戏回放

为了让游戏的双方了解下棋的整个过程,网络五子棋模块设计了游戏回放功能。当游戏结束时,用户可以通过游戏回放了解整个下棋的过程,分析对方下棋的思路,总结经验。

五子棋游戏模块设计了一个存储下棋步骤的双向队列,要实现游戏回放功能,只需要把队列中记录的下棋步骤(每一个队列元素保存了下棋的每一步的棋谱)从头演示一遍就可以了。但是要注意,每个步骤需要停顿一秒,给玩家一个分析的时间。程序关键代码如下:

```

private void backplayToggleButtonActionPerformed(java.awt.event.ActionEvent
    evt) {
    // 如果游戏进行中,提示用户游戏结束后再观看游戏回放
    if (gobangPanell.isStart()) {
        JOptionPane.showMessageDialog(this, "请在游戏结束后,观看游戏回放。");
        backplayToggleButton.setSelected(false);
        return;
    }
    if (!backplayToggleButton.isSelected()) {
        backplayToggleButton.setText("游戏回放");
    } else {
        backplayToggleButton.setText("终止回放");
        new Thread() { // 开启新的线程播放游戏记录
            @Override
            public void run() {
                Object[] toArray = gobangPanell.getOldRec();
                if (toArray == null) {
                    JOptionPane.showMessageDialog(CheessPanel.this, "没有游戏记录",
                        "游戏回放",
                        JOptionPane.WARNING_MESSAGE);
                    backplayToggleButton.setText("游戏回放");
                    backplayToggleButton.setSelected(false);
                    return;
                }
                // 清除界面的结局文字,包括对方胜利、你胜利了、此战平局
                gobangPanell.setTowardsWin(false);
                gobangPanell.setWin(false);
                gobangPanell.setDraw(false);
                for (int i = toArray.length - 1; !gobangPanell.isStart() &&
                    backplayToggleButton.isSelected() &&
                    i >= 0; i--) {
                    try {
                        Thread.sleep(1000); // 线程休眠1秒
                    } catch (InterruptedException ex) {
                        Logger.getLogger(CheessPanel.class.getName()).log(Level.
                            SEVERE, null, ex);
                    }
                }
            }
        };
    }
}

```





```
    }  
    // 根据游戏记录每一步棋谱  
    GobangModel.getInstance().updateChessmanArray((byte[][])  
        toArray[i]);  
    gobangPanel1.repaint();           // 重绘棋盘  
    }  
    backplayToggleButton.setSelected(false);  
    backplayToggleButton.setText("游戏回放");  
    }  
    }.start();  
}
```

## 15.3 游戏登录界面

### 15.3.1 功能概述

主窗体的登录界面是五子棋模块的开始，它的主要功能不是验证用户名与密码，而是定义自己游戏时的昵称和对方主机的 IP 地址。昵称将显示在游戏界面中，包括自己的和对家的昵称。IP 地址是确定对家的唯一条件，只有确定双方的 IP 地址，并且双方都运行了五子棋模块之后，才能进行游戏。登录界面如图 15.13 所示。

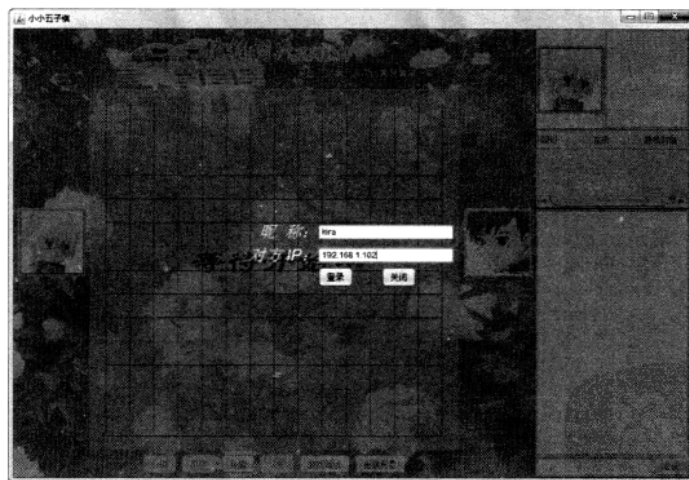


图 15.13 登录联机的程序界面



434



### 15.3.2 绘制登录界面背景

绘制背景时重写 `paintComponent()` 方法，在方法中获取 Java2D 的绘图对象，备份绘图的合成模式，然后设置新的 80% 透明的合成模式，并使用矩形填充整个登录面板。程序关





键代码如下:

```
protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;           // 获取 2D 绘图上下文
    Composite composite = g2.getComposite();    // 备份合成模式
    // 设置绘图使用透明合成规则
    g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.8f));
    g2.fillRect(0, 0, getWidth(), getHeight()); // 使用当前颜色填充矩形空间
    g2.setComposite(composite);                 // 恢复原有合成模式
    super.paintComponent(g2);                  // 执行超类的组件绘制方法
}
```

### 15.3.3 增加窗体控件

登录界面中使用文本框接收用户输入的昵称和对方 IP, 使用标签标示文本框作用, 使用按钮响应用户登录、关闭操作。这些控件使用网格布局, 其关键代码如下:

```
java.awt.GridBagConstraints gridBagConstraints;

jLabel1 = new javax.swing.JLabel();
nameTextField = new javax.swing.JTextField();
jLabel2 = new javax.swing.JLabel();
ipTextField = new javax.swing.JTextField();
loginButton = new javax.swing.JButton();
closeButton = new javax.swing.JButton();

jLabel1.setFont(new java.awt.Font("楷体_gbk", 2, 24));
jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("昵 称: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.anchor = java.awt.GridBagConstraints.EAST;
add(jLabel1, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
gridBagConstraints.ipady = -5;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.insets = new java.awt.Insets(3, 0, 3, 0);
add(nameTextField, gridBagConstraints);

jLabel2.setFont(new java.awt.Font("楷体_gbk", 2, 24));
jLabel2.setForeground(java.awt.Color.white);
jLabel2.setText("对方 IP: ");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
add(jLabel2, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.ipady = -5;
gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 2;
gridBagConstraints.insets = new java.awt.Insets(3, 0, 3, 0);
```





```
add(ipTextField, gridBagConstraints);

loginButton.setText("登录");
loginButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        loginButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.insets = new Insets(0, 0, 0, 40);
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
add(loginButton, gridBagConstraintss);

closeButton.setText("关闭");
closeButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        closeButtonActionPerformed(evt);
    }
});

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.insets = new Insets(0, 0, 0, 55);
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 2;
add(closeButton, gridBagConstraintss);
```

### 15.3.4 处理“登录”按钮事件

编写“登录”按钮的事件处理方法，在该方法中接收用户的昵称和对方主机信息，使用昵称和本机 IP 地址创建本地用户对象发送给对方主机。使用对方主机 IP 地址创建 Socket 连接对象，实现互联操作。程序关键代码如下：

```
private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // 获取主窗体的实例对象
        MainFrame mainFrame = (MainFrame) getParent().getParent();
        String name = nameTextField.getText(); // 获取用户昵称
        if (name.trim().isEmpty()) {
            JOptionPane.showMessageDialog(this, "请输入昵称");
            return;
        }
        String ipText = ipTextField.getText(); // 获取对家 IP 地址
        if (ipText == null || ipText.isEmpty()) {
            JOptionPane.showMessageDialog(this, "请输入对家 IP 地址");
            return;
        }
        ipTextField.setEditable(true);
        InetAddress ip = InetAddress.getByAddress(ipText);
        if (ip.equals(InetAddress.getLocalHost())) {
            JOptionPane.showMessageDialog(this, "不能输入自己的 IP 地址");
            return;
        }
        socket = new Socket(ip, 9528); // 创建 Socket 连接对家主机
        if (socket.isConnected()) { // 如果连接成功
```







```

user = new UserBean(); // 创建用户对象
Time time = new Time(System.currentTimeMillis()); // 获取当前时间对象
user.setName(name); // 初始化用户昵称
user.setHost(InetAddress.getLocalHost()); // 初始化用户 IP
user.setTime(time); // 初始化用户登录时间
socket.setOOBInline(true); // 启用紧急数据的接收
mainFrame.setSocket(socket); // 设置主窗体的 Socket 连接对象
mainFrame.setUser(user); // 添加本地用户对象到主窗体对象
mainFrame.send(user); // 发送本地用户对象到对家主机
setVisible(false); // 隐藏登录窗体
}
} catch (UnknownHostException ex) {
    Logger.getLogger(LoginPanel.class.getName()).log(Level.SEVERE, null, ex);
    JOptionPane.showMessageDialog(this, "输入的 IP 不正确");
} catch (IOException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(this, "对方主机无法连接");
}
}

```

## 15.4 游戏主窗体

### 15.4.1 功能概述

游戏的主窗体包括下棋面板、用户信息面板、用户列表面板和聊天面板等,界面的运行效果如图 15.14 所示。本节将重点讲述用户信息面板、用户列表面板和聊天面板的实现过程,下一节将讲述下棋面板的实现过程。



图 15.14 游戏主窗体效果





## 15.4.2 聊天面板实现

实现聊天面板的关键步骤如下。

(1) 编写 `setSocket()` 方法, 该方法曾在登录面板中使用过, 用户用来设置联机的 `Socket` 对象, 但是, 该方法同时也初始化了 `objout` 对象输出流, 它用于发送字符串对象或其他对象到对家主机。程序关键代码如下:

```
public void setSocket(Socket chatSocketArg) {
    try {
        socket = chatSocketArg;
        OutputStream os = socket.getOutputStream(); // 获取 Socket 的输出流
        objout = new ObjectOutputStream(os); // 创建对象输出流
    } catch (IOException ex) {
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

(2) 编写 `send()` 方法, 该方法用于发送信息到对家主机, 这个信息可以是文本字符串对象也可以是其他类型的对象。在主类中调用该方法发送聊天的文本字符串信息, 但是在其他面板类中, 调用该方法发送用户对象、游戏指令、棋盘信息等内容, 所以该方法的参数是 `Object` 类型。程序关键代码如下:

```
public void send(Object message) {
    try {
        objout.writeObject(message); // 向对象输出流添加对象
        objout.flush();
    } catch (IOException ex) {
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

(3) 编写聊天面板的“发送”按钮的事件处理方法, 该方法将获取用户输入的聊天字符串, 并把该字符串的内容追加到聊天记录的文本区域组件中, 然后调用 `send()` 方法将聊天信息发送给对家。程序关键代码如下:

```
private void sendButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String message = chatTextField.getText(); // 获取文本信息
    if (message == null || message.isEmpty()) {
        return;
    }
    chatTextField.setText(""); // 清空文本框内容
    appendMessage(user.getName() + ": " + message); // 将发送的信息添加到聊天记录
    send(message); // 发送信息
}
```

(4) 编写 `appendMessage()` 方法, 该方法用于向聊天面板的文本区域组件追加换行的聊天信息。程序关键代码如下:

```
protected void appendMessage(final String message) {
    Runnable runnable = new Runnable() { // 创建线程对象
        @Override
        public void run() {
            chatArea.append("\n" + message); // 向聊天文本区域组件追加换行文本
        }
    };
}
```



438







```

        if (SwingUtilities.isEventDispatchThread()) {
            runnable.run(); // 在事件队列线程中执行该线程对象
        } else {
            SwingUtilities.invokeLater(runnable);
        }
    }
}

```

(5) 编写启动 Socket 服务器的 startServer()方法, 该方法将创建 ServerSocket 类的实例对象, 该对象用户接收远程用户的连接。程序关键代码如下:

```

public void startServer() {
    try {
        // 创建 Socket 服务器对象
        final ServerSocket chatSocketServer = new ServerSocket(9528);
        new ReceiveThread(chatSocketServer, this).start(); // 创建接收信息的线程
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(this, "本程序禁止重复运行, 只能同时存在一个实例。",
            "你敢重复运行?", JOptionPane.ERROR_MESSAGE);

        System.exit(0);
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

### 15.4.3 实现用户信息面板与列表面板

实现用户信息面板与列表面板的步骤如下。

(1) 编写 setUser()方法, 该方法用于设置本地用户信息, 包括游戏下棋面板的本地用户昵称、用户列表中的本地用户和用户信息面板的内容等。程序关键代码如下:

```

public void setUser(UserBean user) {
    this.user = user;
    // 向用户信息面板添加昵称
    userInfoTextArea.setText("昵称: " + user.getName() + "\n");
    userInfoTextArea.append("IP: " + user.getHost().getHostAddress() + "\n");
    // 添加 IP 信息
    // 获取用户信息表格组件的数据模型对象
    DefaultTableModel model = (DefaultTableModel) userInfoTable.getModel();
    Vector dataVector = model.getDataVector();
    Vector row = new Vector(); // 使用用户信息创建单行数据的向量
    row.add(user.getName());
    row.add(user.getHost().getHostName());
    row.add(user.getTime());
    if (!dataVector.contains(row)) {
        model.getDataVector().add(row); // 把用户信息添加到表格组件中
    }
    chessPanel1.leftInfoLabel.setText(user.getName()); // 设置本地用户的昵称
    userInfoTable.revalidate();
}

```

(2) 编写 setTowardsUser()方法, 该方法与 setUser()方法功能类似, 但是它用于设置对家的信息。程序关键代码如下:

```

public void setTowardsUser(UserBean user) {
    this.towardsUser = user; // 对家用户对象
    // 获取用户信息列表的表格数据模型
    DefaultTableModel model = (DefaultTableModel) userInfoTable.getModel();
    Vector row = new Vector(); // 创建承载表格单行数据的向量集合对象
}

```





```
row.add(towardsUser.getName()); // 添加用户姓名
row.add(towardsUser.getHost().getHostName()); // 添加主机名称
row.add(towardsUser.getTime()); // 添加用户登录时间
Vector dataVector = model.getDataVector();
if (!dataVector.contains(row)) {
    model.getDataVector().add(row); // 添加用户信息到表格中
}
// 设置对家用户头像的昵称
chessPanel1.rightInfoLabel.setText(towardsUser.getName());
userInfoTable.revalidate();
}
```

## 15.5 下棋面板

### 15.5.1 功能概述

下棋面板用于游戏的控制，包括游戏的开始、悔棋、和棋、认输、清屏、更换游戏背景图等，它还负责在游戏开始时，为双方玩家分配棋子颜色等业务，程序界面如图 15.15 所示。

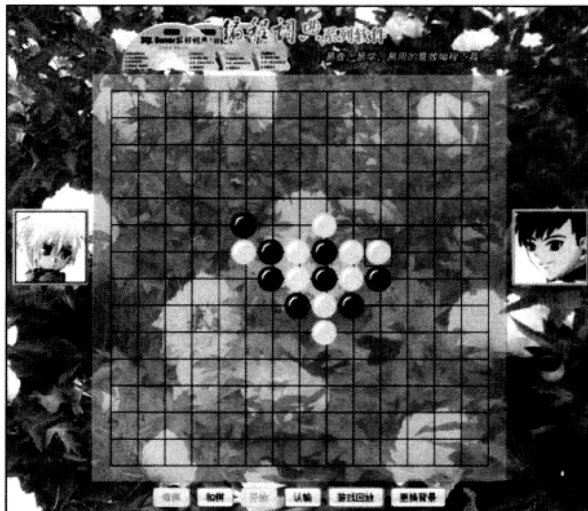


图 15.15 下棋面板



### 15.5.2 实现广告标题栏

游戏面板包含一个广告标题栏，它位于棋盘的上方，如图 15.16 所示。





图 15.16 游戏界面的广告标题栏

当单击该广告栏时,相应的事件监听器会调用 `bannerLabelMouseClicked()` 方法,该方法会调用 `Desktop` 类的 `browse()` 方法使用本地的浏览器打开编程词典网站。程序关键代码如下:

```
private void bannerLabelMouseClicked(java.awt.event.MouseEvent evt) {
    try {
        // 调用 Desktop 类的 browse() 方法浏览编程词典首页
        if (Desktop.isDesktopSupported()) {
            Desktop.getDesktop().browse(new URL("http://www.mrbccd.com").toURI());
        } else {
            JOptionPane.showMessageDialog(this, "当前系统不支持该操作");
        }
    } catch (Exception ex) {
        Logger.getLogger(ChessPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

### 15.5.3 绘制下棋面板中的图片

下棋面板中包括多种图片资源,例如,棋盒图片、背景图片等,其实现绘制过程如下。

(1) 在构造方法中初始化双方棋盒的图片和背景图片对象。另外,该构造方法负责调用初始化界面的 `initComponents()` 方法完成界面布局。程序关键代码如下:

```
public ChessPanel() {
    // 初始化白棋棋盒图片
    WHITE_CHESS_ICON = new javax.swing.ImageIcon(getClass().getResource("/res/whiteChess.png"));
    // 初始化黑棋棋盒图片
    BLACK_CHESS_ICON = new javax.swing.ImageIcon(getClass().getResource("/res/blackChess.png"));
    URL url = getClass().getResource("/res/bg/1.jpg");
    backImg = new ImageIcon(url).getImage(); // 初始化背景图片
    initComponents(); // 调用初始化界面的方法
}
```

(2) 重写父类的 `paintComponent()` 方法,在方法中绘制游戏的背景图片,从而定义新的组件界面。程序关键代码如下:

```
protected void paintComponent(Graphics g) {
    g.drawImage(backImg, 0, 0, getWidth(), getHeight(), null); // 绘制背景图片
}
```

(3) 编写设置棋盒颜色的 `setChessColor()` 方法,该方法用于设置自己和对家头像下方的棋盒颜色。程序关键代码如下:

```
public void setChessColor(ImageIcon color) {
    myChessColorLabel.setIcon(color); // 设置本地用户的棋盒图标
    if (color.equals(WHITE_CHESS_ICON)) { // 设置白棋
        gobangPanel1.setMyColor(GobangPanel.WHITE_CHESSMAN);
    }
}
```





```
towardsChessColorLabel.setIcon(BLACK_CHESS_ICON);
} else if (color.equals(BLACK_CHESS_ICON)) { // 设置黑棋
    gobangPanel1.setMyColor(GobangPanel.BLACK_CHESSMAN);
    towardsChessColorLabel.setIcon(WHITE_CHESS_ICON);
}
revalidate();
}
```

### 15.5.4 下棋前的准备工作

下棋前的准备工作包括刷屏动画、分配棋子等，其实现步骤如下。

(1) 编写 `fillChessBoard()` 方法，该方法用于实现开始游戏时的刷屏动画，根据方法的参数决定使用哪个颜色的棋子填充棋盘，但最终调用该方法的代码位置会再次调用该方法，使用参数 0 清除棋盘上的所有棋子。程序关键代码如下：

```
private void fillChessBoard(final byte chessman) {
    try {
        Runnable runnable = new Runnable() { // 创建刷屏的动画线程
            @Override
            public void run() {
                byte[][] chessmanArray = GobangModel.getInstance().
                    getChessmanArray(); // 获取棋盘数组
                for (int i = 0; i < chessmanArray.length; i += 2) {
                    try {
                        Thread.sleep(10); // 动画间隔时间
                    } catch (InterruptedException ex) {
                        Logger.getLogger(ChessPanel.class.getName()).log(Level.
                            SEVERE, null, ex);
                    }
                    // 使用指定颜色的棋子填充数组的一列
                    Arrays.fill(chessmanArray[i], chessman);
                    Arrays.fill(chessmanArray[(i + 1) % 15], chessman);
                    GobangModel.getInstance().updateChessmanArray
                        (chessmanArray); // 更新棋盘上的棋子
                    // 立即重绘指定区域的棋盘
                    gobangPanel1.paintImmediately(0, 0, getWidth(), getHeight());
                }
            }
        };
        // 在事件队列中执行刷屏
        if (SwingUtilities.isEventDispatchThread()) {
            runnable.run();
        } else {
            SwingUtilities.invokeLaterAndWait(runnable);
        }
    } catch (Exception ex) {
        Logger.getLogger(ChessPanel.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

(2) 编写 `fenqi()` 方法，该方法用于分配双方玩家棋子的颜色，区分规则是先开始的玩家使用白色棋子。程序关键代码如下：

```
private void fenqi() {
    MainFrame frame = (MainFrame) getRootPane().getParent(); // 获取主窗体对象
    // 获取对家开始游戏的时间
```







```

long towardsTime = frame.getTowardsUser().getTime().getTime();
long meTime = frame.getUser().getTime().getTime(); // 获取自己开始游戏的时间
// 根据两个玩家开始游戏时间的先后, 分配棋子的颜色
if (meTime >= towardsTime) {
    frame.getChessPanell().setChessColor(ChessPanel.WHITE_CHESS_ICON);
    frame.getChessPanell().getGobangPanell().setTurn(true);
} else {
    frame.getChessPanell().setChessColor(ChessPanel.BLACK_CHESS_ICON);
    frame.getChessPanell().getGobangPanell().setTurn(false);
}
}

```

(3) 编写 setTurn()方法, 该方法用于设置自己走棋的权限, 如果没有走棋权限的话, 不能在棋盘上任何位置落子。程序关键代码如下:

```

public void setTurn(boolean turn) {
    if (turn) {
        myChessColorLabel.setVisible(true); // 如果获得走棋权利
        towardsChessColorLabel.setVisible(false); // 显示棋盒
    } else {
        myChessColorLabel.setVisible(false); // 隐藏对家棋盒
        towardsChessColorLabel.setVisible(true); // 否则
    }
}

```

(4) 编写 repentOperation()方法, 该方法用于执行悔棋动作, 它首先从记录下棋动作的队列中取出两个走棋的记录 (包含对家走棋和自己走棋), 然后使用队列顶层的当前棋局更新棋盘上的棋子实现悔棋动作。程序关键代码如下:

```

public synchronized void repentOperation() {
    Deque<byte[][]> chessQueue = gobangPanell.getChessQueue(); // 获取下棋队列
    if (chessQueue.isEmpty()) {
        return;
    }
    // 获取上两次走棋的棋谱
    for (int i = 0; i < 2 && !chessQueue.isEmpty(); i++) {
        chessQueue.pop(); // 废弃走棋步骤
    }
    if (chessQueue.size() < 1) {
        chessQueue.push(new byte[15][15]);
    }
    byte[][] pop = chessQueue.peek();
    GobangModel.getInstance().updateChessmanArray(pop); // 更新棋盘的棋子布局
    repaint();
}

```

### 15.5.5 游戏控制面板按钮事件

游戏控制面板包括悔棋、和棋、开始、认输等功能, 其实现步骤如下。

(1) 编写“悔棋”按钮的事件处理方法, 该方法将向对家发送一个悔棋命令的编码, 并使该按钮 5 秒钟内处于禁用的状态。程序关键代码如下:

```

private void backButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 如果没到自己走棋, 提示用户
    if (!gobangPanell.isTurn()) {

```





```
JOptionPane.showMessageDialog(this, "没到你走棋呢。", "请等待...",
JOptionPane.WARNING_MESSAGE);
return;
}
send(OPRATION_REPENT); // 发送“悔棋”命令
new Thread() { // 开启新的线程，使悔棋按钮禁用 5 秒
    @Override
    public void run() {
        try {
            backButton.setEnabled(false);
            sleep(5000);
            backButton.setEnabled(true);
        } catch (InterruptedException ex) {
            Logger.getLogger(ChessPanel.class.getName()).log(Level.SEVERE,
            null, ex);
        }
    }
}.start();
}
```

(2) “和棋”按钮的事件处理方法将向对家发送和棋指令的编码，并使“和棋”按钮在 5 秒钟内不得使用。程序关键代码如下：

```
private void heqiButtonActionPerformed(java.awt.event.ActionEvent evt) {
    send(OPRATION_DRAW); // 发送和棋指令
    new Thread() { // 开启新的线程使“和棋”按钮 5 秒不可用
        @Override
        public void run() {
            try {
                heqiButton.setEnabled(false);
                sleep(5000);
                heqiButton.setEnabled(true);
            } catch (InterruptedException ex) {
                Logger.getLogger(ChessPanel.class.getName()).log(Level.SEVERE,
                null, ex);
            }
        }
    }.start();
}
```

(3) 编写“开始”按钮的事件处理方法，该方法将初始化游戏状态并发送开始命令到对家，然后分配上方棋子的颜色，清除棋盘的棋子。程序关键代码如下：

```
private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 获取主窗体对象
    MainFrame mainFrame = (MainFrame) getRootPane().getParent();
    if (mainFrame.serverSocket == null) {
        JOptionPane.showMessageDialog(this, "请等待对方连接。");
        return;
    }
    if (gobangPanel1.isStart()) {
        return;
    }
    // 设置各个按钮的可用状态
    startButton.setEnabled(false);
    giveupButton.setEnabled(true);
    heqiButton.setEnabled(true);
    backButton.setEnabled(true);
    gobangPanel1.setStart(true); // 设置游戏的开始状态
    gobangPanel1.setTowardsWin(false); // 设置对家胜利状态
}
```







```

gobangPanell.setWin(false);           // 设置自己胜利状态
gobangPanell.setDraw(false);          // 设置和棋状态
send(OPRATION_START);                 // 发送开始指令
fenqi();                              // 分配双方棋子
fillChessBoard(gobangPanell.getMyColor()); // 使用自己的棋子颜色清屏
fillChessBoard((byte) 0);             // 使用空棋子清屏
byte[][] data = new byte[15][15];     // 创建一个空的棋盘布局
GobangModel.getInstance().setChessmanArray(data); // 设置棋盘使用空布局
}

```

(4) 编写“认输”按钮的事件处理方法, 该方法向对家发送一个认输的指令编码, 并启动一个线程使“认输”按钮在 5 秒钟内处于禁用状态。程序关键代码如下:

```

private void giveupButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // 如果没到自己走棋, 提示用户等待
    if (!gobangPanell.isTurn()) {
        JOptionPane.showMessageDialog(this, "没到你走棋呢。", "请等待...",
            JOptionPane.WARNING_MESSAGE);
        return;
    }
    send(OPRATION_GIVEUP); // 发送认输指令
    // 启动一个新的线程使认输按钮 5 秒不可用
    new Thread() {
        @Override
        public void run() {
            try {
                giveupButton.setEnabled(false);
                sleep(5000);
                giveupButton.setEnabled(true);
            } catch (InterruptedException ex) {
                Logger.getLogger(ChessPanel.class.getName()).log(Level.SEVERE,
                    null, ex);
            }
        }
    }.start();
}

```

(5) 游戏回放功能可以观看游戏的比赛记录, 该功能将每间隔 1 秒钟根据棋谱的记录演示双方的游戏过程。“游戏回放”按钮的事件处理方法由 backplayToggleButtonActionPerformed() 方法实现, 程序关键代码如下:

```

private void backplayToggleButtonActionPerformed(java.awt.event.ActionEvent
    evt) {
    // 如果游戏进行中, 提示用户游戏结束后再观看游戏回放
    if (gobangPanell.isStart()) {
        JOptionPane.showMessageDialog(this, "请在游戏结束后, 观看游戏回放。");
        backplayToggleButton.setSelected(false);
        return;
    }
    if (!backplayToggleButton.isSelected()) {
        backplayToggleButton.setText("游戏回放");
    } else {
        backplayToggleButton.setText("终止回放");
        new Thread() { // 开启新的线程播放游戏记录
            @Override
            public void run() {
                Object[] toArray = gobangPanell.getOldRec();
                if (toArray == null) {
                    JOptionPane.showMessageDialog(ChessPanel.this, "没有游戏记录",
                        "游戏回放",

```





```
        JOptionPane.WARNING_MESSAGE);
        backplayToggleButton.setText("游戏回放");
        backplayToggleButton.setSelected(false);
        return;
    }
    // 清除界面的结局文字, 包括对方胜利、你胜利了、此战平局
    gobangPanell.setTowardsWin(false);
    gobangPanell.setWin(false);
    gobangPanell.setDraw(false);
    for (int i = toArray.length - 1; !gobangPanell.isStart() &&
        backplayToggleButton.isSelected() && i >= 0; i--) {
        try {
            Thread.sleep(1000); // 线程休眠 1 秒
        } catch (InterruptedException ex) {
            Logger.getLogger(ChessPanel.class.getName()).log(Level.
                SEVERE, null, ex);
        }
        // 根据游戏记录每一步棋谱
        GobangModel.getInstance().updateChessmanArray((byte[][] )
            toArray[i]);
        gobangPanell.repaint(); // 重绘棋盘
    }
    backplayToggleButton.setSelected(false);
    backplayToggleButton.setText("游戏回放");
}
}.start();
}
```

(6) “更换背景”按钮的事件监听器由 `ButtonActionListener` 类定义, 它实现了 `ActionListener` 接口和接口的 `actionPerformed()` 方法, 并在该方法中重新定义背景图片对象 `backImg`, 然后调用 `repaint()` 方法使用新的背景绘制界面。程序关键代码如下:

```
private class ButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(final ActionEvent e) {
        backIndex = backIndex % 9 + 1; // 获取 9 张背景图片的索引的递增
        URL url = getClass().getResource("/res/bg/" + backIndex + ".jpg");
        backImg = new ImageIcon(url).getImage(); // 初始化棋盘图片
        repaint(); // 重新绘制下棋面板
    }
}
```

## 15.6 棋盘面板

### 15.6.1 功能概述

棋盘面板和下棋面板是密不可分的, 这两个面板共同完成下棋功能。棋盘面板负责五子棋游戏的落子和游戏规则, 如图 15.17 所示。





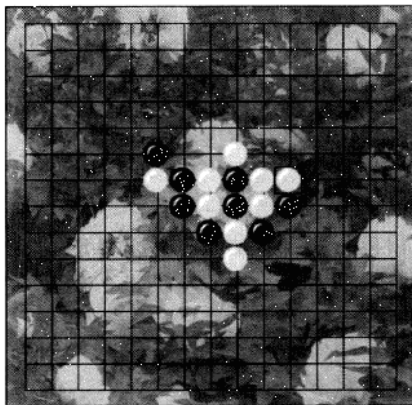


图 15.17 棋盘面板

### 15.6.2 绘制棋盘面板

实现棋盘面板的关键步骤如下。

(1) 继承 JPanel 类编写 GobangPanel 棋盘面板组件。在 GobangPanel 类中定义开始、胜利、和棋等游戏状态的控制变量、记录游戏过程的队列变量等，并在构造方法中初始化棋盘面板需要的图片对象。程序关键代码如下：

```
public class GobangPanel extends javax.swing.JPanel {
    private static final long serialVersionUID = 777731592731933312L;
    // 黑白棋盒的图标对象和星号的图像对象，以及背景图片对象
    private Image white_chessman_img;
    private Image black_chessman_img;
    private Image rightTop_img;
    int chessWidth, chessHeight; // 棋子宽度与高度
    public final static byte WHITE_CHESSMAN = 1;
    public final static byte BLACK_CHESSMAN = -1;
    Dimension size; // 棋盘面板的大小
    private boolean start = false; // 开始
    private Object[] oldRec;
    Deque<byte[][]> chessQueue = new LinkedList(); // 游戏的队列记录
    private boolean turn = false; // 是否轮到自己走棋
    private boolean towardsWin; // 对方胜利
    private boolean win; // 胜利
    private boolean draw; // 和棋
    private ChessPanel chessPanel;

    public GobangPanel() {
        URL white_url = getClass().getResource("/res/whiteChessman.png");
        URL black_url = getClass().getResource("/res/blackChessman.png");
        URL rightTop_url = getClass().getResource("/res/rightTop.gif");
        // 初始化白棋图片
        white_chessman_img = new ImageIcon(white_url).getImage();
        // 初始化黑棋图片
        black_chessman_img = new ImageIcon(black_url).getImage();
        // 初始化连成线的棋子上的星图
        rightTop_img = new ImageIcon(rightTop_url).getImage();
    }
}
```





```
size = new Dimension(getWidth(), getHeight());
setPreferredSize(size);
initComponents();
}
// 省略部分代码
}
```

(2) 重写父类的 `paint()` 方法, 该方法负责绘制组件的界面, 本模块在棋盘面板的 `paint()` 方法中绘制棋盘、棋子和游戏的提示信息。绘制棋子时又包含多种落子状态, 例如, 分别绘制白棋、黑棋, 绘制最近白棋或黑棋的落子 (带边框的)、胜利后为连成线的棋子绘制五星等。程序关键代码如下:

```
public void paint(Graphics g1) {
    Graphics2D g = (Graphics2D) g1;
    super.paint(g); // 调用父类的绘图方法
    if (chessPanel != null) {
        chessPanel.setTurn(turn);
    }
    Composite composite = g.getComposite(); // 备份合成模式
    drawPanel(g); // 调用绘制棋盘的方法
    g.translate(4, 4);
    size = new Dimension(getWidth(), getHeight()); // 获取棋盘面板的大小
    chessWidth = size.width / 15; // 初始化棋子宽
    chessHeight = size.height / 15; // 初始化棋子高
    byte[][] chessmanArray = gobangModel1.getChessmanArrayCopy();
    for (int i = 0; i < chessmanArray.length; i++) { // 遍历棋盘数据模型绘制棋子
        for (int j = 0; j < chessmanArray[i].length; j++) {
            byte chessman = chessmanArray[i][j];
            int x = i * chessWidth;
            int y = j * chessHeight;
            if (chessman != 0)
                System.out.println("chess is:" + chessman);
            if (chessman == WHITE_CHESSMAN) { // 绘制白棋
                g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
            } else if (chessman == BLACK_CHESSMAN) { // 绘制黑棋
                g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
            } else if (chessman == (WHITE_CHESSMAN ^ 3)) { // 绘制最近的白棋落子
                g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
                g.drawRect(x, y, chessWidth, chessHeight);
            } else if (chessman == (BLACK_CHESSMAN ^ 3)) { // 绘制最近的黑棋落子
                g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
                g.drawRect(x, y, chessWidth, chessHeight);
            } // 绘制导致胜利的连线白棋
            } else if (chessman == ((byte) (WHITE_CHESSMAN ^ 8))) {
                g.drawImage(white_chessman_img, x, y, chessWidth, chessHeight, this);
                g.drawImage(rightTop_img, x, y, chessWidth, chessHeight, this);
            } else if (chessman == (BLACK_CHESSMAN ^ 8)) { // 绘制导致胜利的连线黑棋
                g.drawImage(black_chessman_img, x, y, chessWidth, chessHeight, this);
                g.drawImage(rightTop_img, x, y, chessWidth, chessHeight, this);
            }
        }
    }
}

if (!isStart()) { // 如果游戏未处于开始状态
    // 如果游戏处于胜利或和棋状态, 绘制棋盘提示信息
    if (towardsWin || win || draw) {
        g.setComposite(AlphaComposite.SrcOver.derive(0.7f)); // 设置 70%
        // 透明的合成规则
        String mess = "对方胜利"; // 定义提示信息
        g.setColor(Color.RED); // 设置前景色为红色
    }
}
```





```

        if (win) {
            mess = "你胜利了";
            g.setColor(new Color(0x007700));
        } else if (draw) {
            mess = "此战平局";
            g.setColor(Color.YELLOW);
        }
        // 设置提示文本的字体为隶书、粗斜体、大小 72
        Font font = new Font("隶书", Font.ITALIC | Font.BOLD, 72);
        g.setFont(font);

        // 获取字体渲染上下文对象
        FontRenderContext context = g.getFontRenderContext();
        // 计算提示信息的文本所占用的像素空间
        Rectangle2D stringBounds = font.getStringBounds(mess, context);
        double fontWidth = stringBounds.getWidth(); // 获取提示文本的宽度
        g.drawString(mess, (int) ((getWidth() - fontWidth) / 2), getHeight() / 2); // 居中绘制提示信息
        g.setComposite(composite); // 恢复原有合成规则
        // 如果当前处于其他未开始游戏的状态
    } else {
        String mess = "等待开始...";
        Font font = new Font("隶书", Font.ITALIC | Font.BOLD, 48);
        g.setFont(font);
        FontRenderContext context = g.getFontRenderContext();
        Rectangle2D stringBounds = font.getStringBounds(mess, context);
        double fontWidth = stringBounds.getWidth(); // 获取提示文本的宽度
        g.drawString(mess, (int) ((getWidth() - fontWidth) / 2), getHeight() / 2); // 居中绘制提示文本
    }
}

```

(3) 编写绘制棋盘的 `drawPanel()` 方法, 该方法将利用半透明合成规则绘制透明背景的棋盘, 另外, 为了更好地支持棋盘缩放功能, 使用户能自由调整棋盘大小, 这里使用 `drawLine()` 画线的方法绘制了棋盘的网格。程序界面如图 15.18 所示。

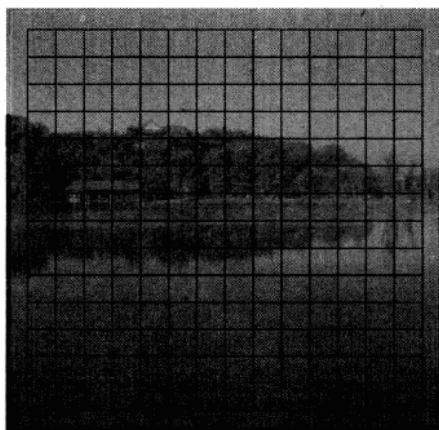


图 15.18 绘制半透明的棋盘效果

绘制棋盘的关键代码如下:

```
private void drawPanel(Graphics2D g) {
```



```
Composite composite = g.getComposite();           // 备份合成规则
Color color = g.getColor();                       // 备份前景颜色
g.setComposite(AlphaComposite.SrcOver.derive(0.6f)); // 设置透明合成
g.setColor(new Color(0xAABBAA));                  // 设置前景白色
g.fillRect(0, 0, getWidth(), getHeight(), true); // 绘制半透明的矩形
g.setComposite(composite);                        // 恢复合成规则
g.setColor(color);                                // 恢复原来前景色
int w = getWidth();                              // 棋盘宽度
int h = getHeight();                             // 棋盘高度
int chessW = w / 15, chessH = h / 15;            // 棋子宽度和高度
int left = chessW / 2 + (w % 15) / 2;            // 棋盘左边界
int right = left + chessW * 14;                  // 棋盘右边界
int top = chessH / 2 + (h % 15) / 2;             // 棋盘上边界
int bottom = top + chessH * 14;                   // 棋盘下边界
for (int i = 0; i < 15; i++) {
    // 画每条横线
    g.drawLine(left, top + (i * chessH), right, top + (i * chessH));
}
for (int i = 0; i < 15; i++) {
    // 画每条竖线
    g.drawLine(left + (i * chessW), top, left + (i * chessW), bottom);
}
```

### 15.6.3 实现游戏规则算法

arithmetic()方法是根据五子棋的游戏规则编写的计算方法，它根据当前棋子的类型和位置，计算是黑棋赢还是白棋赢，计算方法是落子点为中心，向左右两边、上下两边、正斜两边、反斜两边查找同一类型的棋子，如果棋子数大于等于5，则表示当前下棋者为赢家。游戏胜利和失败的界面分别如图 15.19 和图 15.20 所示。

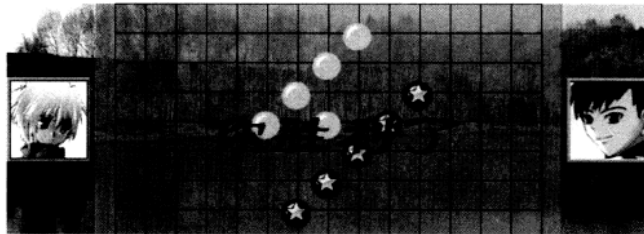


图 15.19 游戏胜利界面

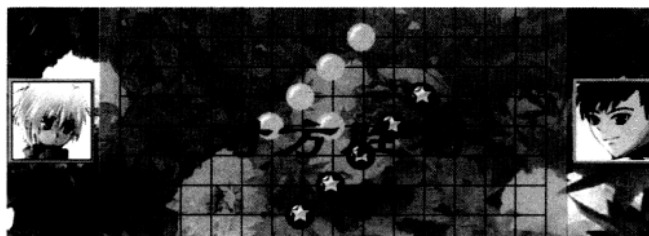


图 15.20 游戏失败界面







实现五子棋算法的程序关键代码如下：

```
public int arithmetic(int n, int Arow, int Acolumn) {
    int n3 = n ^ 3;
    byte n8 = (byte) (n ^ 8);
    byte[][] note = gobangModell.getChessmanArrayCopy();
    int BCount = 1;
    // 纵向查找
    boolean Lbol = true;
    boolean Rbol = true;
    BCount = 1;
    for (int i = 1; i <= 5; i++) {
        if ((Acolumn + i) > 14) { // 如果棋子超出最大列数
            Rbol = false;
        }
        if ((Acolumn - i) < 0) { // 如果棋子超出最小列数
            Lbol = false;
        }
        if (Rbol == true) {
            if (note[Arow][Acolumn + i] == n || note[Arow][Acolumn + i] == n3) {
                // 如果横向向右有相同的棋子
                ++BCount;
                note[Arow][Acolumn + i] = n8;
            } else {
                Rbol = false;
            }
        }
        if (Lbol == true) {
            if (note[Arow][Acolumn - i] == n || note[Arow][Acolumn - i] == n3) {
                // 如果横向向左有相同的棋子
                ++BCount;
                note[Arow][Acolumn - i] = n8;
            } else {
                Lbol = false;
            }
        }
        if (BCount >= 5) { // 如果同类型的棋子数大于等于5个
            note[Arow][Acolumn] = n8;
            gobangModell.updateChessmanArray(note);
            repaint();
            return n; // 返回胜利一方的棋子
        }
    }

    // 横向查找
    note = gobangModell.getChessmanArrayCopy();
    boolean Ubol = true;
    boolean Dbol = true;
    BCount = 1;
    for (int i = 1; i <= 5; i++) {
        if ((Arow + i) > 14) { // 如果超出棋盘的最大行数
            Dbol = false;
        }
        if ((Arow - i) < 0) { // 如果超出棋盘的最小行数
            Ubol = false;
        }
        if (Dbol == true) {
            if (note[Arow + i][Acolumn] == n || note[Arow + i][Acolumn] == n3) {
                // 如果向上有同类型的棋子
                ++BCount;
                note[Arow + i][Acolumn] = n8;
            } else {
                Dbol = false;
            }
        }
        if (Ubol == true) {
            if (note[Arow - i][Acolumn] == n || note[Arow - i][Acolumn] == n3) {
                // 如果向下有同类型的棋子
                ++BCount;
                note[Arow - i][Acolumn] = n8;
            } else {
                Ubol = false;
            }
        }
    }
}
```







```
        if (note[Arow - i][Acolumn] == n || note[Arow - i][Acolumn] == n3)
        {
            // 如果向下有同类型的棋子
            ++BCount;
            note[Arow - i][Acolumn] = n8;
        } else {
            Ubol = false;
        }
    }
    if (BCount >= 5) {
        // 如果同类型的棋子大于等于 5 个
        note[Arow][Acolumn] = n8;
        gobangModel1.updateChessmanArray(note);
        repaint();
        return n;
        // 返回胜利一方的棋子
    }
}
// 正向查找
note = gobangModel1.getChessmanArrayCopy();
boolean LUbol = true;
boolean RDbol = true;
BCount = 1;
for (int i = 1; i <= 5; i++) {
    if ((Arow - i) < 0 || (Acolumn - i < 0)) { // 如果超出左面的斜线
        LUbol = false;
    }
    if ((Arow + i) > 14 || (Acolumn + i > 14)) { // 如果超出右面的斜线
        RDbol = false;
    }
    if (LUbol == true) {
        // 如果左上斜线上有相同类型的棋子
        if (note[Arow - i][Acolumn - i] == n || note[Arow - i][Acolumn - i] == n3) {
            ++BCount;
            note[Arow - i][Acolumn - i] = n8;
        } else {
            LUbol = false;
        }
    }
    if (RDbol == true) {
        // 如果右下斜线上有相同类型的棋子
        if (note[Arow + i][Acolumn + i] == n || note[Arow + i][Acolumn + i] == n3) {
            ++BCount;
            note[Arow + i][Acolumn + i] = n8;
        } else {
            RDbol = false;
        }
    }
    if (BCount >= 5) {
        // 如果同类型的棋子大于等于 5 个
        note[Arow][Acolumn] = n8;
        gobangModel1.updateChessmanArray(note);
        repaint();
        return n;
        // 返回胜利一方的棋子
    }
}
// 反斜查找
note = gobangModel1.getChessmanArrayCopy();
boolean RUBol = true;
boolean LDbol = true;
BCount = 1;
for (int i = 1; i <= 5; i++) {
    if ((Arow - i) < 0 || (Acolumn + i > 14)) {
        RUBol = false;
    }
    if ((Arow + i) > 14 || (Acolumn - i < 0)) {
        LDbol = false;
    }
    if (RUBol == true) {
        // 如果左下斜线上有相同类型的棋子
        if (note[Arow - i][Acolumn + i] == n || note[Arow - i][Acolumn + i] == n3) {
```





```

        ++BCount;
        note[Arow - i][Acolumn + i] = n8;
    } else {
        RUBol = false;
    }
}
if (LDbol == true) {
    // 如果右上斜线上有相同类型的棋子
    if (note[Arow + i][Acolumn - i] == n || note[Arow + i][Acolumn - i] == n3) {
        ++BCount;
        note[Arow + i][Acolumn - i] = n8;
    } else {
        LDbol = false;
    }
}
if (BCount >= 5) { // 如果同类型的棋子大于等于5个
    note[Arow][Acolumn] = n8;
    gobangModel1.updateChessmanArray(note);
    repaint();
    return n; // 返回胜利一方的棋子
}
}
return 0;
}

```

#### 15.6.4 编写棋盘模型

棋盘模型是一个JavaBean,它主要用于记录棋盘上的棋子。如果该模型的数据被改变,那么它将通知所有监听该模型的事件监听器,棋盘面板就定义了一个这样的事件监听器。程序关键代码如下:

```

gobangModel1 = new GobangModel(); // 创建棋盘模型的实例对象
// 为棋盘添加事件监听器
gobangModel1.addPropertyChangeListener(new PropertyChangeListener() {
    @Override
    public void propertyChange(java.beans.PropertyChangeEvent evt) {
        gobangModel1PropertyChange(evt); // 调用事件处理方法
    }
});

```

在这个事件监听器中将调用 gobangModel1PropertyChange()方法执行相应的业务逻辑,该方法将最新的棋盘数据压入队列中,然后根据新的棋盘数据重新绘制棋盘界面上的棋子,程序关键代码如下:

```

private void gobangModel1PropertyChange(java.beans.PropertyChangeEvent evt) {
    chessQueue.push(gobangModel1.getChessmanArrayCopy()); // 将新的棋盘布局压入队列
    repaint(); // 重回棋盘界面
}

```

在定义棋盘模型的JavaBean时,必须定义一个PropertyChangeSupport类的实例对象作为类的字段(全局变量),并将各种事件监听器的工作委托给它。程序关键代码如下:

```

public class GobangModel extends Object implements Serializable {
    private static final long serialVersionUID = 5022354839275938047L;
    private PropertyChangeSupport propertySupport; // 定义属性工具类
    private static GobangModel model; // 定义自身的变量
    private byte[][] chessmanArray = new byte[15][15]; // 定义棋子数组
    // 定义属性名称
    public static final String PROP_CHESSMANARRAY = "chessmanArray";
}

```





```
public static GobangModel getInstance() {
    if (model == null) {
        model = new GobangModel();
    }
    return model;
}

public GobangModel() {
    propertySupport = new PropertyChangeSupport(this); // 初始化属性工具栏
    model = this;
}

public byte[][] getChessmanArray() {
    return chessmanArray; // 返回棋子数组
}

public void setChessmanArray(byte[][] chessmanArray) {
    this.chessmanArray = chessmanArray;
    // 报告所有已注册监听器的绑定属性更新
    propertySupport.firePropertyChange(PROP_CHESSMANARRAY, null, chessmanArray);
}

public synchronized void updateChessmanArray(byte[][] chessmanArray) {
    this.chessmanArray = chessmanArray;
}

public void addPropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.addPropertyChangeListener(listener); // 添加事件监听器
}

public void removePropertyChangeListener(PropertyChangeListener listener) {
    // 移除事件监听器
    propertySupport.removePropertyChangeListener(listener);
}

byte[][] getChessmanArrayCopy() {
    byte[][] newArray = new byte[15][15]; // 创建一个二维数组
    for (int i = 0; i < newArray.length; i++) {
        // 复制数组
        newArray[i] = Arrays.copyOf(chessmanArray[i], newArray[i].length);
    }
    return newArray;
}
}
```

### 15.6.5 编写联机通信类

本模块使用一个联机通信类来接收对家发送的所有信息，包括游戏命令代码的接收与处理。这个联机通信类 `ReceiveThread` 是一个线程类，它继承 `Thread` 类并重写类的 `run()` 方法，来处理联机业务。`run()` 方法是线程的核心方法，本类在该方法中接收远程计算机的联机请求，根据对方的联机信息填充登录面板的 IP 地址文本框，从网络中读取 Java 对象，并根据对象的类型判断信息的种类是聊天信息、登录信息还是命令代码等，并做相应的业务处理。程序关键代码如下：

```
public class ReceiveThread extends Thread {
    private final ServerSocket chatSocketServer;
    MainFrame frame;
    private String host;
```







```

@Override
public void run() {
    while (true) {
        try {
            frame.serverSocket = chatSocketServer.accept(); // 接收 Socket 连接
            Socket serverSocket = frame.serverSocket;
            // 获取对方主机信息
            host = serverSocket.getInetAddress().getHostName();
            // 获取对方 IP 地址
            String ip = serverSocket.getInetAddress().getHostAddress();
            // 询问是否接受联机
            int link = JOptionPane.showConfirmDialog(frame, "收到" + host + "
            的联机请求, 是否接受?");
            if (link == JOptionPane.YES_OPTION) { // 如果接受联机
                // 获取登录面板的实例
                LoginPanel loginPanel = (LoginPanel) frame.getRootPane().getGlassPane();
                loginPanel.setLinkIp(ip); // 设置登录面板的对家 IP 信息
            }
            serverSocket.setOOBInline(true); // 启用紧急数据的接收
            // 获取网络输入流
            InputStream is = serverSocket.getInputStream();
            // 创建对象输入流
            ObjectInputStream objis = new ObjectInputStream(is);
            while (frame.isVisible()) {
                serverSocket.sendUrgentData(255); // 发送紧急数据
                // 从对象输入流读取 Java 对象
                Object messageObj = objis.readObject();
                // 如果读取的对象是 String 类型
                if (messageObj instanceof String) {
                    // 获取对家昵称
                    String name = frame.getTowardsUser().getName();
                    // 将字符串信息添加到通信面板
                    frame.appendMessage(name + ": " + messageObj);
                } // 如果读取的是字节数组对象
                else if (messageObj instanceof byte[][]) {
                    // 将数组对象设置为棋盘模型数据
                    GobangModel.getInstance().setChessmanArray(
                        (byte[][] messageObj);
                    frame.getChessPanel1().getGobangPanel1().setTurn(true);
                    // 获得走棋权限
                    // 获取自己的棋子颜色
                    byte myColor = frame.getChessPanel1().getGobangPanel1().
                    getMyColor();
                    frame.getChessPanel1().getGobangPanel1().zhengliBoard
                    (myColor); // 整理棋盘
                    // 梅棋按钮可用
                    frame.getChessPanel1().backButton.setEnabled(true);
                } else if (messageObj instanceof Integer) { // 如果是整型对象
                    oprationHandler(messageObj); // 命令代码的接收和处理方法
                } // 如果是用户实体对象
                else if (messageObj instanceof UserBean) {
                    UserBean user = (UserBean) messageObj;
                    frame.setTowardsUser(user); // 设置对家信息
                }
            }
        } catch (SocketException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE,
            null, ex);
            JOptionPane.showMessageDialog(frame, "连接中断");
            frame.getChessPanel1().reInit();
            DefaultTableModel model = (DefaultTableModel) frame.userInfoTable
            getTableModel();
            model.setRowCount(0);
            frame.getGlassPane().setVisible(true);
        } catch (IOException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE,
            null, ex);
        }
    }
}

```







```
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE,
            null, ex);
    }
}

public ReceiveThread(ServerSocket chatSocketServer, MainFrame outer) {
    super();
    this.frame = outer;
    this.chatSocketServer = chatSocketServer;
}

private void operationHandler(Object messageObj) {
    int code = (Integer) messageObj; // 获取命令代码
    String towards = frame.getTowardsUser().getName(); // 获取对家昵称
    int option;
    switch (code) {
        case ChessPanel.OPRATION_REPENT: // 如果是悔棋请求
            System.out.println("请求悔棋");
            // 询问玩家是否同意对方悔棋
            option = JOptionPane.showConfirmDialog(frame, towards + "要悔棋,
                是否同意?", "求你了,我走错了,让我悔棋!!!", JOptionPane.YES_NO_OPTION,
                JOptionPane.INFORMATION_MESSAGE);
            // 在聊天面板添加悔棋信息
            frame.appendMessage("对方请求悔棋.....");
            if (option == JOptionPane.YES_OPTION) { // 如果同意悔棋
                // 发送同意悔棋的消息
                frame.send(ChessPanel.OPRATION_NODE_REPENT);
                // 执行本地的悔棋操作
                frame.getChessPanel().repentOperation();
                // 添加悔棋信息到聊天面板
                frame.appendMessage("接受对方的悔棋请求.");
                frame.send(frame.getUser().getName() + "接受悔棋请求");
            } else { // 如果不同意悔棋
                // 添加不同意悔棋的信息到聊天面板
                frame.send(frame.getUser().getName() + "拒绝悔棋请求");
                frame.appendMessage("拒绝了对方的悔棋请求.");
            }
            break;
        case ChessPanel.OPRATION_NODE_REPENT: // 如果是同意悔棋命令
            System.out.println("同意悔棋命令");
            frame.getChessPanel().repentOperation(); // 执行本地的悔棋操作
            frame.appendMessage("悔棋成功"); // 把悔棋成功信息添加到聊天面板
            break;
        case ChessPanel.OPRATION_NODE_DRAW: // 如果是同意和棋命令
            System.out.println("同意和棋命令");
            // 设置和棋状态为 true
            frame.getChessPanel().getGobangPanel().setDraw(true);
            frame.getChessPanel().reInit(); // 初始化游戏状态变量
            frame.appendMessage("此战平局."); // 将和棋信息添加到聊天面板
            break;
        case ChessPanel.OPRATION_DRAW: // 如果是和棋请求
            System.out.println("请求和棋");
            // 询问玩家是否同意和棋
            option = JOptionPane.showConfirmDialog(frame, towards + "请求和
                棋,是否同意?", "大哥,和棋吧!!!", JOptionPane.YES_NO_OPTION, J
                OptionPane.QUESTION_MESSAGE);
            frame.appendMessage("对方请求和棋....."); // 添加信息到聊天面板
            if (option == JOptionPane.YES_OPTION) { // 如果同意和棋
                // 发送接受和棋的消息
                frame.send(ChessPanel.OPRATION_NODE_DRAW);
                // 设置和棋状态为 true
                frame.getChessPanel().getGobangPanel().setDraw(true);
                frame.getChessPanel().reInit(); // 初始化游戏状态变量
                // 添加信息到聊天面板
                frame.appendMessage("接受对方的和棋请求.");
            }
        }
    }
}
```



```

        frame.send(frame.getUser().getName() + "接受和棋请求");
    } else {
        // 发送拒绝信息
        frame.send(frame.getUser().getName() + "拒绝和棋请求");
        frame.appendMessage("拒绝对方的和棋请求。");
    }
    break;
case ChessPanel.OPERATION_GIVEUP:
    System.out.println("对方认输");
    // 询问玩家是否同意对方认输
    option = JOptionPane.showConfirmDialog(frame, towards + "请求认输, 是否同意?", "对方认输", JOptionPane.YES_NO_OPTION);
    frame.appendMessage("对方请求认输.....");
    if (option == JOptionPane.YES_OPTION) {
        frame.send(ChessPanel.WIN);
        // 设置胜利状态为 true
        frame.getChessPanel1().getGobangPanel1().setWin(true);
        frame.getChessPanel1().reInit();
        frame.appendMessage("接受对方的认输请求。");
    } else {
        frame.send(frame.getUser().getName() + "拒绝认输请求");
        frame.appendMessage("拒绝对方的认输请求。");
    }
    break;
case ChessPanel.OPERATION_START:
    System.out.println("请求开始");
    // 如果自己已经执行游戏开始动作
    if (frame.getChessPanel1().getGobangPanel1().isStart()) {
        // 发送全部开始命令
        frame.send(ChessPanel.OPERATION_ALL_START);
        // 设置对家游戏开始状态为 true
        frame.getChessPanel1().setTowardsStart(true);
    }
    break;
case ChessPanel.OPERATION_ALL_START:
    System.out.println("回应开始请求");
    // 设置对家为开始状态
    frame.getChessPanel1().setTowardsStart(true);
    break;
case ChessPanel.WIN:
    System.out.println("对方胜利");
    // 设置对家胜利状态为 true
    frame.getChessPanel1().getGobangPanel1().setTowardsWin(true);
    frame.getChessPanel1().reInit();
    break;
default:
    System.out.println("未知操作代码: " + code);
}
}
}

```



# 第二篇

## 项目实战篇

本篇主要内容：

- 第 16 章 酒店管理系统
- 第 17 章 企业人事管理系统
- 第 18 章 医药综合管理系统（内容参见光盘）
- 第 19 章 进销存管理系统（内容参见光盘）

书山有路勤为径  
学海无涯苦作舟



# 第 16 章

---

## 酒店管理系统

( Swing+SQL Server 2005 实现 )

随着计算机技术的不断发展，应用软件已经遍及社会的各行各业。大到厂矿校企，小到餐饮洗浴，并且以其独特的优势服务于社会。传统的餐饮业采用手工记账的方式，不仅费时费力，还容易出现错误。使用酒店管理系统，管理员只需要轻点几下鼠标和键盘，就可以轻松完成这些任务。既提高了工作效率，又节省了人力资源，为餐饮业的快速发展创造了巨大的空间。

通过本章的内容，读者可以学习到：

- » 酒店管理系统的软件结构和业务流程
- » 根据用户输入的部分内容快速获取商品的方法
- » 人性化控制商品数量的方法
- » 系统自动结账的实现方法
- » Swing 中鼠标或键盘事件的使用方法
- » 年、月、日下拉列表框联动保证日期合法性的方法
- » 通过正则表达式验证用户输入数据合法性的方法
- » 系统时钟的实现方法



## 16.1 开发背景

“民以食为天”，随着人们生活水平的不断提高，餐饮业在服务行业中的地位也越来越重要，如何从激烈的竞争中脱颖而出，已经成为每位餐饮经营者在思考的问题。

经过多年的发展，对餐饮企业的管理已经逐渐由简单的人工管理，逐步进入到规范、科学管理的阶段。众所周知，在科学管理的具体实现方法中，最有效的工具就是应用管理软件进行管理。

以往的人工操作管理中存在着许多问题，例如：

- ☐ 人工计算账单容易出现错误。
- ☐ 收银工作中容易发生账单丢失。
- ☐ 客人具体消费信息难以查询。
- ☐ 无法对以往营业数据进行查询。

## 16.2 系统分析

随着餐饮行业的迅速发展，现有人工管理方式已不能满足管理者的需求，广大餐饮业经营者已经意识到使用计算机应用软件的重要性，决定在餐饮企业的经营管理上引入计算机应用软件管理系统。

根据餐饮行业的特点和实际情况，酒店管理系统应以餐饮业务为基础，突出前台管理，重视营业数据分析等功能，从专业角度出发，努力为餐饮管理者提供科学、有效的管理模式和数据分析功能。

开台点菜功能是酒店管理系统的最常用功能，也是酒店管理系统的主要功能之一，所以需要将该功能设计得更加人性化和智能化，例如，在确定添加菜品时，既可以通过菜品编号确定，又可以通过菜品助记码确定，并且默认添加菜品的数量为一个。

自动结账功能也是酒店管理系统的最常用功能，同样是酒店管理系统的主要功能之一，用户只需要选中结账的台号，系统就会自动为选中的台号计算消费金额，并且用户输入实收金额后，系统还会自动计算出需要找零的金额，这样既节省了系统操作员的精力，又避免了由于计算失误造成的损失。

报表功能是酒店管理系统不可缺少的一部分，因为酒店管理系统是一个记账式软件，如果一个记账式软件没有报表功能，就好比一个双目失明的狮子，已经失去了生存的能力，对于一个餐饮企业，日报表是不可缺少的。

再就是系统安全和系统维护功能，这是应用软件必不可少的一部分，用来保障软件的安全运行。







## 16.3 系统设计

### 16.3.1 系统目标

依据餐饮行业的特点，本系统需要实现以下目标：

- ☐ 操作简单方便、界面简洁大方。
- ☐ 方便、快捷的开台点菜功能。
- ☐ 智能化定位菜品的功能。
- ☐ 快速查看开台点菜信息的功能。
- ☐ 自动结账功能。
- ☐ 按开台和商品实现的日结账功能。
- ☐ 按日消费额汇总统计实现的月结账功能。
- ☐ 按日营业额实现的年结账功能。
- ☐ 系统运行稳定、安全可靠。

### 16.3.2 系统功能结构

酒店管理系统分为前台服务、后台管理、结账报表、系统安全几个大的功能，在每个大的功能下面各自又有独自的小功能，如前台服务里有开台点菜、智能化获取菜品、自动结账，后台管理有台号管理、菜系管理、菜品管理，等等。详细的结构如图 16.1 所示。

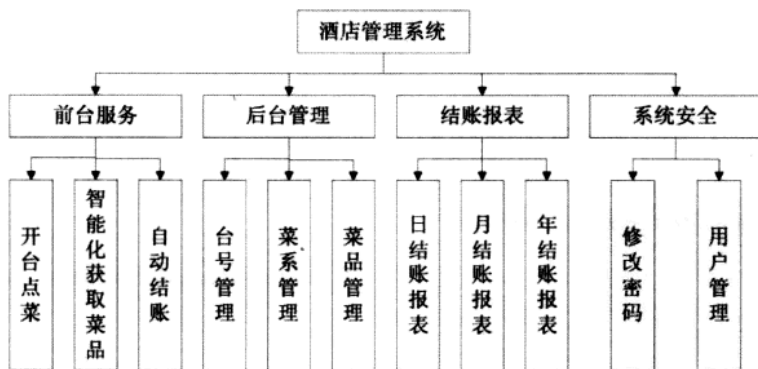


图 16.1 酒店管理系统功能结构

### 16.3.3 系统预览

酒店管理系统由多个程序界面组成，下面仅列出几个典型界面，其他界面效果请参见





光盘中的源程序。

酒店管理系统的主窗体效果如图 16.2 所示，窗体的中间部分用来显示当前的开台及点菜信息，窗体的下方用来操作该系统，例如，开台点菜，自动结账，台号、菜系和菜品的维护，营业额报表等。

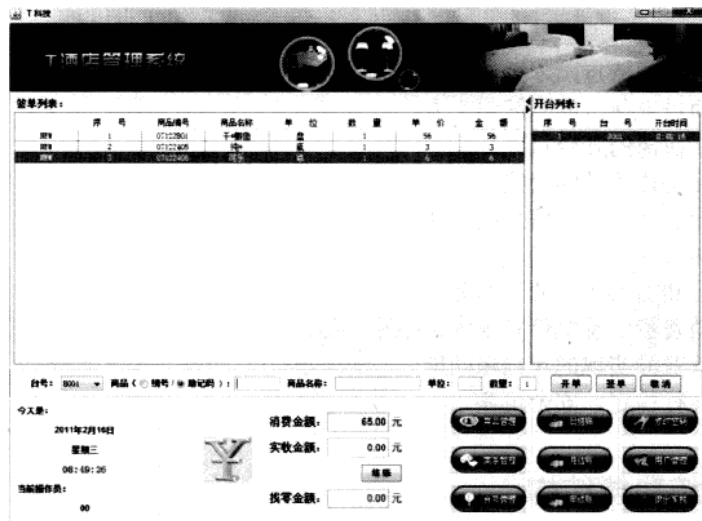


图 16.2 酒店管理系统主窗体效果

单击图 16.2 中右下方的“台号管理”按钮，将打开如图 16.3 所示的“台号管理”对话框，该对话框用来维护台号信息，包括台号及座位数。

单击图 16.2 中右下方的“菜品管理”按钮，将打开如图 16.4 所示的“菜品管理”对话框，该对话框用来维护菜品信息，包括名称、助记码、菜系、单位和单价，其中，助记码用来在点菜时快速获取菜品信息，建议设置为菜品名称的首字母，例如，将菜品“红烧狮子头”的助记码设置为“hsszt”。

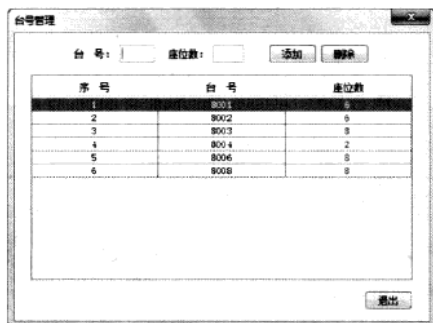


图 16.3 “台号管理”对话框

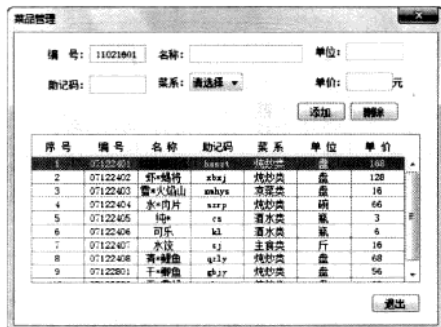


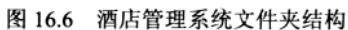
图 16.4 “菜品管理”对话框

单击图 16.2 中右下方的“日结账”按钮，将打开如图 16.5 所示的“日结账”对话框，该对话框用来统计指定日期的销售情况，包括日营业额和各个商品的日营业额。

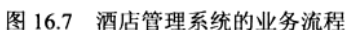


图 16.5 “日结账”对话框

每个项目都会有相应的文件夹组织结构。当项目中的窗体过多时,为了便于查找和使用,可以将窗体进行分类,放入不同的文件夹中,这样既便于前期开发,又便于后期维护。酒店管理系统文件夹组织结构如图 16.6 所示。



酒店管理系统的业务流程如图 16.7 所示。





## 16.4 数据库设计

在开发应用程序时，对数据库的操作是必不可少的，而一个数据库的设计优秀与否，将直接影响到软件的开发进度和性能，所以对数据库的设计就显得尤为重要。数据库的设计要根据程序的需求及其功能制订，如果在开发软件之前不能很好地设计数据库，在开发过程中将反复修改数据库，必将严重影响开发进度。

### 16.4.1 数据库概要说明

酒店管理系统的需求包括开台点菜功能、智能化获取菜品功能、自动结账功能、营业额报表功能等。在这些功能中主要涉及的数据表包括台号表、菜品表、消费单表；为了使系统更完善，还需要为菜品分类，即需要用到菜系表；为了实现菜品的日销售情况统计，还要建立一个消费项目表，用来记录消费单消费的菜品种类。

### 16.4.2 数据库概念设计

数据库设计是系统设计过程中的重要组成部分，它是通过管理系统的整体需求而设计的，数据库设计的好坏直接影响到系统的后期开发。下面对本系统中具有代表性的数据库设计进行详细说明。

餐台和菜系在本系统中最简单的实体，在本系统中用来描述餐台信息的只有台号和座位数，而描述菜系的主要是名称。餐台信息表（tb\_dest 表）的 E-R 图如图 16.8 所示，菜系信息表（tb\_sort 表）的 E-R 图如图 16.9 所示。



图 16.8 餐台信息表 E-R 图



图 16.9 菜系信息表 E-R 图

在描述菜品实体时，加入了助记码，目的是为了实现智能化获取菜品功能，通过这一功能系统操作员可以快速地获取顾客所点的菜品信息。菜品信息表（tb\_menu 表）的 E-R 图如图 16.10 所示。



图 16.10 菜品信息表 E-R 图





消费单 (tb\_order\_form 表) 用来记录每次消费的相关信息, 例如消费时使用的餐台、消费时间、消费金额等。消费单信息表的 E-R 图如图 16.11 所示。



图 16.11 消费单信息表 E-R 图

消费项目 (tb\_order\_item 表) 用来记录每个消费单消费的菜品, 记录的主要信息有消费单、消费菜品的名称、消费数量、消费额。消费项目信息表的 E-R 图如图 16.12 所示。

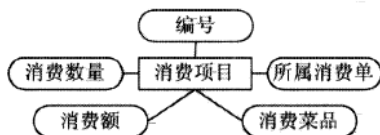


图 16.12 消费项目信息表 E-R 图

### 16.4.3 数据库逻辑设计

在 SQL Server 2005 数据库中, 创建名为 db\_DrinkeryManagel 的数据库, 然后在数据库中根据数据表的 E-R 图创建数据表。其中主要表结构如下所示。

#### 1. 餐台信息表 tb\_dest 的结构 (如表 16.1 所示)。

表 16.1 餐台信息表 tb\_dest

字段名称	数据类型	字段大小	说明
num	varchar	5	台号
seating	int		座位数

#### 2. 菜品信息表 tb\_menu 的结构 (如表 16.2 所示)。

表 16.2 菜品信息表 tb\_menu

字段名称	数据类型	字段大小	说明
num	char	8	编号
sort_id	int		所属菜系
name	varchar	20	名称
code	varchar	10	主机名
unit	varchar	4	单位
unit_price	int		单价
state	char	4	状态



3. 消费单 tb\_order\_form 的结构（如表 16.3 所示）。

表 16.3 消费单 tb\_order\_form

字段名称	数据类型	字段大小	说明
num	char	11	编号
desk_num	varchar	5	消费餐台
datetime	datetime		消费时间
money	int		消费金额
user_id	int		操作用户

4. 消费项目表 tb\_order\_item 的结构（如表 16.4 所示）。

表 16.4 消费项目表 tb\_order\_item

字段名称	数据类型	字段大小	说明
id	int		编号
order_for_num	char	11	所属消费单
menu_num	char	8	消费菜品
amount	int		消费数量
total	int		消费额

5. 菜系信息表 tb\_sort 的结构（如表 16.5 所示）。

表 16.5 菜系信息表 tb\_sort

字段名称	数据类型	字段大小	说明
id	int		序号
name	varchar	20	名称

## 16.5 公共模块设计

### 16.5.1 编写数据库连接类



数据库连接类负责加载数据库驱动程序，以及创建和关闭数据库连接，为了最大程度地应用每个已经创建的数据库连接，这里将其保存到了 ThreadLocal 类的对象中。

466

首先，在数据库连接类中定义一些常量，包括连接数据库使用的驱动程序、连接数据库的路径、连接数据库使用的用户名和密码，并且定义一个 ThreadLocal 类的对象，用来保存已经创建的数据库连接。具体代码如下：

```
private static final String DRIVERCLASS =  
"com.microsoft.sqlserver.jdbc.SQLServerDriver";
```





```
private static final String URL = "jdbc:sqlserver://localhost:1433;
DatabaseName=db_DrinkeryManagel";
private static final String USERNAME = "sa";
private static final String PASSWORD = "";
```

然后,编写用来加载数据库驱动程序的代码,通常情况下将其放到静态代码块中,这样做的好处是只在该类第一次被加载(即第一次被调用)时执行加载数据库驱动程序的动作,避免了反复加载数据库驱动程序,从而提高软件的性能。具体代码如下:

```
static { // 通过静态方法加载数据库驱动
    try {
        Class.forName(DRIVERCLASS).newInstance(); // 加载数据库驱动
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

最后,编写用来创建和关闭数据库连接的方法,这里将这两个方法均定义为静态的,这样通过类名就可以调用方法,方便使用。在这两个方法中首先从 ThreadLocal 类的对象中获得数据库连接,然后判断是否存在可用的数据库连接,如果存在则直接返回或关闭,否则重新创建。具体代码如下:

```
public static Connection getConnection() { // 创建数据库连接的方法
    Connection conn = threadLocal.get(); // 从线程中获得数据库连接
    if (conn == null) { // 没有可用的数据库连接
        try {
            // 创建新的数据库连接
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            threadLocal.set(conn); // 将数据库连接保存到线程中
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return conn;
}

public static boolean closeConnection() { // 关闭数据库连接的方法
    boolean isClosed = true;
    Connection conn = threadLocal.get(); // 从线程中获得数据库连接
    threadLocal.set(null); // 清空线程中的数据库连接
    if (conn != null) { // 数据库连接可用
        try {
            conn.close(); // 关闭数据库连接
        } catch (SQLException e) {
            isClosed = false;
            e.printStackTrace();
        }
    }
    return isClosed;
}
```

## 16.5.2 封装常用的操作数据库的方法

对数据库的操作包括查询、添加、修改和删除,其中查询是通过 executeQuery(String sql) 方法执行 SQL 语句实现的,添加、修改和删除是通过 executeUpdate(String sql) 方法执行 SQL 语句实现的。在本系统中共提供了 4 个用来执行查询的方法,分别用来查询多个记





录、查询指定记录、查询多个记录的指定值和查询指定记录的指定值；F 个用来添加、修改和删除记录的方法。由于篇幅有限，在这里只介绍用来查询多个记录的方法，以及用来添加、修改和删除记录的方法。

下面的方法用来查询多个记录。为了将检索结果直接应用于表格控件，这里全部用 Vector 向量对象封装查询结果，并且为代表每一行的向量添加了行序号。具体代码如下：

```
protected Vector selectSomeNote(String sql) {
    // 创建结果集向量
    Vector<Vector<Object>> vector = new Vector<Vector<Object>>();
    Connection conn = JDBC.getConnection(); // 获得数据库连接
    try {
        Statement stmt = conn.createStatement(); // 创建连接状态对象
        ResultSet rs = stmt.executeQuery(sql); // 执行 SQL 语句获得查询结果
        // 获得查询数据表的列数
        int columnCount = rs.getMetaData().getColumnCount();
        int row = 1; // 定义行序号
        while (rs.next()) { // 遍历结果集
            Vector<Object> rowV = new Vector<Object>(); // 创建行向量
            rowV.add(new Integer(row++)); // 添加行序号
            for (int column = 1; column <= columnCount; column++) {
                rowV.add(rs.getObject(column)); // 添加列值
            }
            vector.add(rowV); // 将行向量添加到结果集向量中
        }
        rs.close(); // 关闭结果集对象
        stmt.close(); // 关闭连接状态对象
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return vector; // 返回结果集向量
}
```

下面的方法用来添加、修改和删除记录。这里采用手动提交，目的是捕获持久化异常，并回退数据库，以保证数据的合法性。具体代码如下：

```
public boolean longHaul(String sql) {
    boolean isLongHaul = true; // 默认持久化成功
    Connection conn = JDBC.getConnection(); // 获得数据库连接
    try {
        conn.setAutoCommit(false); // 设置为手动提交
        Statement stmt = conn.createStatement(); // 创建连接状态对象
        stmt.executeUpdate(sql); // 执行 SQL 语句
        stmt.close(); // 关闭连接状态对象
        conn.commit(); // 提交持久化
    } catch (SQLException e) {
        isLongHaul = false; // 持久化失败
        try {
            conn.rollback(); // 回滚
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    }
    return isLongHaul; // 返回持久化结果
}
```







### 16.5.3 自定义表格控件

在使用表格时, 通常情况下将表格列的值设置为居中显示, 这样看起来更美观。由 `JTable` 控件实现的表格将一个表格分为了两个部分, 一部分是表格头, 即用来显示表格列名的部分, 另一部分是除表格头之外的部分。表格的每一部分对应一个 `DefaultTableCellRenderer` 类的对象, 即单元格对象, 通过该对象可以设置所属表格部分包含单元格的相关属性, 例如, 设置单元格内容的显示位置。

通过 `JTable` 类的 `getDefaultRenderer()` 方法可以得到单元格对象, 这个单元格对象代表的是除表格头之外的部分包含的单元格, 在 `MTable` 类中对该方法进行了简单的重构, 即通过单元格对象设置单元格内容水平为居中显示。具体代码如下:

```
public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {
    // 获得除表格头部分之外的单元格对象
    DefaultTableCellRenderer tableRenderer = (DefaultTableCellRenderer) super.
        getDefaultRenderer(columnClass);
    // 设置单元格内容居中显示
    tableRenderer.setHorizontalAlignment(SwingConstants.CENTER);
    return tableRenderer;
}
```

如果想设置表格头的相关绘制属性, 首先要获得表格头对象, 即 `JTableHeader` 类的对象, 通过 `JTableHeader` 类的 `setReorderingAllowed(boolean reorderingAllowed)` 方法可以设置表格列是否可以重排; 通过 `JTableHeader` 类的 `getDefaultRenderer()` 方法也可以得到单元格对象, 这个单元格对象代表的是表格头包含的单元格, 在 `MTable` 类中对该方法也进行了简单的重构, 即通过单元格对象设置单元格内容 (即列名) 水平为居中显示。具体代码如下:

```
public JTableHeader getTableHeader() {
    JTableHeader tableHeader = super.getTableHeader(); // 获得表格头对象
    tableHeader.setReorderingAllowed(false);           // 设置表格列不可重排
    // 获得表格头的单元格对象
    DefaultTableCellRenderer headerRenderer = (DefaultTableCellRenderer)
        tableHeader.getDefaultRenderer();
    // 设置单元格内容 (即列名) 居中显示
    headerRenderer.setHorizontalAlignment(SwingConstants.CENTER);
    return tableHeader;
}
```

`JTable` 类的 `setRowSelectionInterval(int fromRow, int toRow)` 方法用来设置表格中选中的行, 第一个参数为开始选中行的索引, 第二个参数为结束选中行的索引。在本系统中表格的选择模式为单选, 如果通过该方法设置, 需要设置两个参数, 所以在 `MTable` 类中实现了一个重载方法 `setRowSelectionInterval(int row)`, 用来设置唯一被选中的行, 该方法仍然需要调用前面的方法。具体代码如下:

```
public void setRowSelectionInterval(int row) {           // 通过重载实现自己的方法
    setRowSelectionInterval(row, row);
}
```





## 16.5.4 编写利用正则表达式验证数据合法性的方法

在获得用户的输入内容时,经常需要验证用户输入数据的合法性,例如,对用户输入日期的验证。验证这一类数据时,较好的办法是利用正则表达式,所以本系统提供了一个可重用的利用正则表达式验证数据合法性的方法。每次使用时只需要传入验证规则和验证内容,返回值为 `boolean` 型,当返回 `true` 时表示验证通过,数据合法;当返回 `false` 时表示验证未通过,数据不合法。具体代码如下:

```
public static boolean execute(String rule, String content) {  
    Pattern pattern = Pattern.compile(rule); // 利用验证规则创建 Pattern 对象  
    Matcher matcher = pattern.matcher(content); // 利用验证内容获得 Matcher 对象  
    return matcher.matches(); // 返回验证结果  
}
```

## 16.6 主窗体模块设计

### 16.6.1 主窗体模块概述

本系统将主窗体划分为 6 个工作区,分别是开台签单工作区、自动结账工作区、后台管理工作区、结账报表工作区、系统安全工作区和系统提示区,酒店管理系统主窗体效果如图 16.13 所示。



图 16.13 酒店管理系统主窗体效果





## 16.6.2 主窗体模块技术分析

主窗体模块中用到的主要技术是 JSplitPane, JSplitPane 用于分隔两个 (只能是两个) Component。两个 Component 图形化分隔以外观实现为基础, 并且这两个 Component 可以由用户交互式调整大小。使用 JSplitPane.HORIZONTAL\_SPLIT 可让分隔窗格中的两个 Component 从左到右排列, 或者使用 JSplitPane.VERTICAL\_SPLIT 使其从上到下排列。改变 Component 大小的首选方式是调用 setDividerLocation, 其中, location 是新的 x 或 y 位置, 具体取决于 JSplitPane 的方向。要将 Component 调整到其首选大小, 可调用 resetToPreferredSizes。

当用户调整 Component 的大小时, Component 的最小大小用于确定 Component 能够设置的最大/最小位置。如果两个控件的最小大小大于分隔窗格的大小, 则分隔条将不允许您调整其大小。当用户调整分隔窗格大小时, 新的空间以 resizeWeight 为基础在两个控件之间分配。在默认情况下, 值为 0 表示右边/底部的控件获得所有空间, 而值为 1 表示左边/顶部的控件获得所有空间。

## 16.6.3 主窗体模块实现过程

在开台签单工作区中使用了分割面板, 这样, 系统操作员可以根据实际需要, 调整开台列表和签单列表的大小, 并设置分割面板支持快速展开/折叠的分割条, 效果如图 16.14 所示, 既可以将光标移动到分割条的上方随意调整分割条的位置, 又可以通过单击分割条上的◀或▶按钮将分割条移动到分割面板的最左侧或最右侧, 单击另一个则使分割条恢复到原位置; 不支持快速展开/折叠的分割条效果如图 16.15 所示。

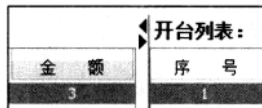


图 16.14 支持快速展开/折叠的分割条

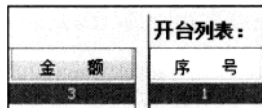


图 16.15 不支持快速展开/折叠的分割条

实现图 16.14 所示的分割面板的关键代码如下:

```
final JSplitPane splitPane = new JSplitPane();           // 创建分割面板对象
splitPane.setOrientation(JSplitPane.HORIZONTAL_SPLIT); // 设置为水平分割
splitPane.setDividerLocation(755);                     // 设置面板默认的分割位置
splitPane.setDividerSize(10);                          // 设置分割条的宽度
splitPane.setOneTouchExpandable(true);                 // 设置为支持快速展开/折叠分割条
splitPane.setBorder(new TitledBorder(null, "", TitledBorder.DEFAULT_
JUSTIFICATION,
TitledBorder.DEFAULT_POSITION, null, null));           // 设置面板的边框
// 将分割面板添加到上级容器中
getContentPane().add(splitPane, BorderLayout.CENTER);
final JPanel leftPanel = new JPanel();                  // 创建放于分割面板左侧的普通面板对象
leftPanel.setLayout(new BorderLayout());                // 设置面板的布局管理器
splitPane.setLeftComponent(leftPanel);                 // 将普通面板对象添加到分割面板的左侧
// 此处省略部分代码
final JPanel rightPanel = new JPanel();                 // 创建放于分割面板右侧的普通面板对象
```



471





```
splitPane.setRightComponent(rightPanel); // 将普通面板对象添加到分割面板的右侧
```

在系统提示区提供了时钟的功能，如图 16.16 所示。

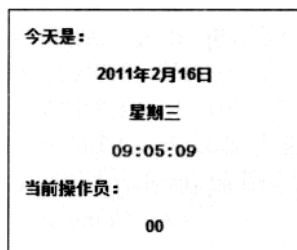


图 16.16 系统提示区的时钟

系统提示区的时钟是显示到标签控件上的，对标签控件的具体设置代码如下：

```
timeLabel = new JLabel(); // 创建用于显示时间的标签对象
// 设置标签中的文字为宋体、粗体、14 号
timeLabel.setFont(new Font("宋体", Font.BOLD, 14));
timeLabel.setForeground(new Color(255, 0, 0)); // 设置标签中的文字为红色
// 设置标签中的文字居中显示
timeLabel.setHorizontalAlignment(SwingConstants.CENTER);
clueOnPanel.add(timeLabel); // 将标签添加到上级容器中
new Time().start(); // 开启线程
```

下面在 TipWizardFrame 类中创建一个内部类 Time，该类继承了线程类 Thread，并重构了 run() 方法，每隔一秒修改一次用于标签中显示的时间信息。具体代码如下：

```
class Time extends Thread { // 创建内部类
    public void run() { // 重构父类的方法
        while (true) {
            Date date = new Date(); // 创建日期对象
            // 获取当前时间，并显示到时间标签中
            timeLabel.setText(date.toString().substring(11, 19));
            try {
                Thread.sleep(1000); // 令线程休眠 1 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

## 16.7 用户登录窗口模块设计



### 16.7.1 用户登录窗口模块概述



用户登录窗口是每一个应用软件都不可缺少的部分，其主要功能是保证用户的数据安全；同时用户登录窗口也是用户看到的第一个系统界面，因此，一个设计优秀的用户登录窗口，将有效地加深用户对系统的第一印象。





用户登录窗口的设计优秀与否，主要包括以下几个方面：

- ☐ 美观大方。
- ☐ 简单易懂。
- ☐ 安全性高。
- ☐ 使用方便。

### 16.7.2 用户登录窗口模块技术分析

要使用户登录界面美观大方，就离不开对图片的使用，但是 `JPanel` 类并不支持绘制背景图片的功能，即将控件绘制到图片的上方，可以通过重写 `JPanel` 类的 `paintComponent(Graphics g)` 方法，实现支持绘制背景图片的功能。具体代码如下：

```
public class MPanel extends JPanel {
    private static final long serialVersionUID = 7056298952360607443L;
    private ImageIcon imageIcon;          // 声明一个图片对象
    public MPanel(URL imgUrl) {
        super();                          // 继承父类的构造方法
        setLayout(new GridBagLayout());   // 将布局管理器修改为网格组布局
        imageIcon = new ImageIcon(imgUrl); // 根据传入的 URL 创建 ImageIcon 对象
        // 设置面板与图片等大
        setSize(imageIcon.getIconWidth(), imageIcon.getIconHeight());
    }
    @Override
    // 重写 JPanel 类的 paintComponent() 方法
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);          // 调用 JPanel 类的 paintComponent() 方法
        // 通过 ImageIcon 对象获得 Image 对象
        Image image = imageIcon.getImage();
        g.drawImage(image, 0, 0, null);    // 绘制 Image 对象，即将图片绘制到面板中
    }
}
```

利用通过继承 `JPanel` 类得到的 `MPanel` 类，就可以很方便地将图片设置为面板的背景图片了。这样，在设计用于用户登录界面的背景图片时，就可以将一些辅助信息设计到图片上了，如图 16.17 所示就是为本系统的用户登录界面设计的背景图片。



图 16.17 为用户登录界面设计的背景图片

利用如图 16.17 所示图片作为用户登录界面的背景图片，在绘制用户登录界面时，就





只需要利用代码绘制 1 个下拉列表框、1 个密码框和 3 个按钮就可以了，绘制完成后的本系统用户登录界面如图 16.18 所示。



图 16.18 用户登录界面

### 16.7.3 用户登录窗口模块实现过程

#### ■ 本模块使用的数据表: tb\_user

首先创建一个用于用户登录界面的窗体，为窗体设置标题、大小等信息，并将一个支持背景图片功能的面板添加到窗体中，具体代码如下：

```
public class LandFrame extends JFrame {
    private static final long serialVersionUID = -3430911625527498847L;
    private JPasswordField passwordField;           // 密码框
    private JComboBox usernameComboBox;             // 用户名下拉列表框
    public LandFrame() {
        // 首先设置窗口的相关信息
        super();                                     // 调用父类的构造方法
        setTitle(" T 科技");                          // 设置窗口的标题
        setResizable(false);                          // 设置窗口不可以改变大小
        setAlwaysOnTop(true);                         // 设置窗口总在最前方
        setBounds(100, 100, 428, 292);               // 设置窗口的大小
        // 设置当关闭窗口时执行的动作
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 下面将创建一个面板对象并添加到窗口的容器中
        final MPanel panel = new MPanel(this.getClass().getResource("/img/land_
background.jpg"));
        panel.setLayout(new GridBagLayout()); // 设置面板的布局管理器为网格组布局
        // 将面板添加到窗体中
        getContentPane().add(panel, BorderLayout.CENTER);
        // 此处省略部分代码
    }
}
```

下面将依次创建一个下拉列表框控件和一个密码框控件，并利用网格组布局管理器将它们添加到背景面板中，分别用于选择登录用户和输入登录密码。具体代码如下：

```
// 创建并设置用户名下拉列表框
usernameComboBox = new JComboBox(); // 创建用户名下拉列表框控件对象
usernameComboBox.setMaximumRowCount(5); // 设置下拉列表框最多可显示的选项数
usernameComboBox.addItem("请选择"); // 为下拉列表框添加提示项
// 为下拉列表框添加事件监听器
usernameComboBox.addActionListener(new UsernameComboBoxActionListener());
// 创建网格组布局管理器对象
```







```
final GridBagConstraints gridBagConstraints = new GridBagConstraints();
gridBagConstraints.anchor = GridBagConstraints.WEST; // 设置为靠左侧显示
gridBagConstraints.gridy = 1; // 设置行索引为 1
gridBagConstraints.gridx = 2; // 设置列索引为 2
// 将控件按指定的布局管理器添加到面板中
panel.add(usernameComboBox, gridBagConstraints);
// 创建并设置密码框
passwordField = new JPasswordField(); // 创建密码框控件对象
passwordField.setColumns(20); // 设置密码框可显示的字符数
passwordField.setText(""); // 设置密码框默认显示 6 个空格
// 为密码框添加焦点监听器
passwordField.addFocusListener(new PasswordFieldFocusListener());
// 创建网格组布局管理器对象
final GridBagConstraints gridBagConstraints_1 = new GridBagConstraints();
// 设置控件外部上方的填充量为 5 像素
gridBagConstraints_1.insets = new Insets(5, 0, 0, 0);
gridBagConstraints_1.anchor = GridBagConstraints.WEST; // 设置为靠左侧显示
gridBagConstraints_1.gridy = 2; // 设置行索引为 2
gridBagConstraints_1.gridx = 2; // 设置列索引为 2
// 将控件按指定的布局管理器添加到面板中
panel.add(passwordField, gridBagConstraints_1);
```

下面将创建一个用来显示按钮的面板,该面板的背景必须为透明的,否则将遮盖背景图片。该面板采用的布局管理器为水平箱布局,并且其占用网格组布局模式的两列。具体代码如下:

```
// 创建并设置一个用来添加 3 个按钮的面板
final JPanel buttonPanel = new JPanel(); // 创建一个用来添加按钮的面板
buttonPanel.setOpaque(false); // 设置面板的背景为透明
// 设置面板采用水平箱布局
buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));
// 创建网格组布局管理器对象
final GridBagConstraints gridBagConstraints_4 = new GridBagConstraints();
// 设置控件外部上方的填充量为 10 像素
gridBagConstraints_4.insets = new Insets(10, 0, 0, 0);
gridBagConstraints_4.gridwidth = 2; // 设置其占两列
gridBagConstraints_4.gridy = 3; // 设置行索引为 3
gridBagConstraints_4.gridx = 1; // 设置列索引为 1
// 将控件按指定的布局管理器添加到面板中
panel.add(buttonPanel, gridBagConstraints_4);
```

用来绘制“登录”、“重置”和“退出”按钮的代码基本相同,所以在这里只介绍用来绘制“登录”按钮的代码。因为按钮的显示内容均是通过图片实现的,所以在创建按钮对象时,建议设置为不绘制按钮的边框,也不绘制按钮的内容区域,目的是使按钮变为透明的,避免在按钮图片的四周出现不希望的绘制效果;建议将按钮边框和标签之间的间隔设置为 0。还可以为不同状态的按钮设置不同效果的图片,如图 16.19 所示为默认情况下按钮的效果,图 16.20 中的“登录”按钮则展示了当光标位于按钮上方时按钮的效果,图 16.21 中的“登录”按钮则展示了当按钮被按下时按钮的效果。

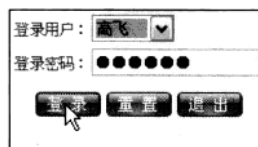
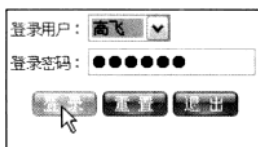
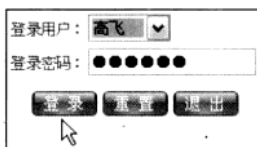


图 16.19 光标不在按钮上方 图 16.20 光标在按钮上方 图 16.21 按钮被按下的瞬间





用来绘制“登录”按钮的具体代码如下：

```
// 创建并设置一个“登录”按钮，并将其添加到用来添加按钮的面板中
final JButton landButton = new JButton(); // 创建“登录”按钮控件对象
landButton.setMargin(new Insets(0, 0, 0, 0)); // 设置按钮边框和标签之间的间隔
landButton.setContentAreaFilled(false); // 设置不绘制按钮的内容区域
landButton.setBorderPainted(false); // 设置不绘制按钮的边框
// 获得默认情况下“登录”按钮显示图片的 URL
URL landUrl = this.getClass().getResource("/img/land_submit.png");
// 设置默认情况下“登录”按钮显示的图片
landButton.setIcon(new ImageIcon(landUrl));
// 获得当鼠标经过登录按钮时显示图片的 URL
URL landOverUrl = this.getClass().getResource("/img/land_submit_over.png");
// 设置当鼠标经过“登录”按钮时显示的图片
landButton.setRolloverIcon(new ImageIcon(landOverUrl));
// 获得当登录按钮被按下时显示图片的 URL
URL landPressedUrl = this.getClass().getResource("/img/land_submit_pressed.png");
// 设置当“登录”按钮被按下时显示的图片
landButton.setPressedIcon(new ImageIcon(landPressedUrl));
// 为“登录”按钮添加事件监听器
landButton.addActionListener(new LandButtonActionListener());
buttonPanel.add(landButton); // 将“登录”按钮添加到用来添加按钮的面板中
```

当第一次使用本系统时，在数据库中将不存在系统管理员。在这种情况下，系统将提供一个默认用户，供用户登录后添加管理员，如图 16.22 所示。添加管理员后，将不再提供系统默认用户，如图 16.23 所示。

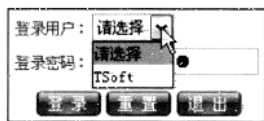


图16.22 系统默认用户



图 16.23 系统管理员

用户单击“登录”按钮后，系统将首先判断是否选择了登录用户，然后再判断是系统默认用户，还是系统管理员，最后验证登录密码，如果通过验证则登录成功，否则登录失败并弹出提示。实现“登录”按钮的具体代码如下：

```
class LandButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获得登录用户的名称
        String username = usernameComboBox.getSelectedItem().toString();
        if (username.equals("请选择")) { // 查看是否选择了登录用户
            JOptionPane.showMessageDialog(null, "请选择登录用户！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE); // 弹出提示
            resetUsernameAndPassword(); // 恢复登录用户和登录密码
        }
        char[] passwords = passwordField.getPassword(); // 获得登录用户的密码
        // 将密码从 char 型数组转换成字符串
        String inputPassword = turnCharsToString(passwords);
        if (username.equals("TSoft")) { // 查看是否为默认用户登录
            if (inputPassword.equals("111")) { // 查看密码是否为默认密码
                land(null); // 登录成功
                String infos[] = { "请立刻单击“用户管理”按钮添加用户！", "添加用户后需要重新登录，本系统才能正常使用！" }; // 组织提示信息
                JOptionPane.showMessageDialog(null, infos, "友情提示",
                    JOptionPane.INFORMATION_MESSAGE); // 弹出提示
            }
        }
    }
}
```







```

        } else {
            // 密码错误
            JOptionPane.showMessageDialog(null, "默认用户'TSoft'的登录密码为
            '111'!", "友情提示", JOptionPane.INFORMATION_MESSAGE); // 弹出提示
            passwordField.setText("111"); // 将密码设置为默认密码
        }
    } else {
        if (inputPassword.length() == 0) { // 用户未输入登录密码
            JOptionPane.showMessageDialog(null, "请输入登录密码!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE); // 弹出提示
            resetUsernameAndPassword(); // 恢复登录用户和登录密码
        }
        // 查询登录用户
        Vector user = Dao.getInstance().sUserByName(username);
        String password = user.get(5).toString(); // 获得登录用户的密码
        if (inputPassword.equals(password)) { // 查看登录密码是否正确
            land(user); // 登录成功
        } else { // 登录密码错误
            JOptionPane.showMessageDialog(null, "登录密码错误, 请确认后重新登录!",
            "友情提示", JOptionPane.INFORMATION_MESSAGE); // 弹出提示
            resetUsernameAndPassword(); // 恢复登录用户和登录密码
        }
    }
}

private void resetUsernameAndPassword() { // 恢复登录用户和登录密码
    usernameComboBox.setSelectedIndex(0); // 恢复选中的登录用户为“请选择”项
    passwordField.setText(" "); // 恢复密码框的默认值为 6 个空格
    return; // 直接返回
}

private void land(Vector user) { // 登录成功
    TipWizardFrame tipWizard = new TipWizardFrame(user); // 创建主窗体对象
    tipWizard.setVisible(true); // 设置主窗体可见
    setVisible(false); // 设置登录窗口不可见
}
}

```

## 16.8 开台签单工作区设计

开台签单工作区是本系统最常用的工作区,所以需要将该工作区设计得更人性化和智能化。例如,在获取欲添加的菜品时,既可以通过菜品编号获得,又可以通过菜品助记码获得,并且默认菜品的数量为一个,因为这是最通用的。

### 16.8.1 开台签单工作区功能概述

开台签单工作区的主要功能有开台、点菜、加菜、签单、查看开台信息和签单信息,开台签单工作区的效果如图 16.24 所示。





图 16.24 开台签单工作区效果

在顾客就餐时，首先在图 16.24 下方的“台号”下拉列表框中选择分配的台号，然后选择菜品名称的选择方式。默认情况下是使用助记码来获取菜品名称。在输入菜品的助记码之后，在“商品名称”文本框中会显示菜品名称，并在“单位”文本框中显示菜品的单位。在“数量”文本框中管理员可以输入菜品的数量，最后单击“开单”按钮完成该菜品的添加操作。当所有菜品添加完成后，可以单击“签单”按钮完成点菜。在点菜过程中，可以使用“取消”按钮取消已经点的菜品。

如果顾客在用餐的过程中要求添加菜品，既可以在“台号”下拉列表框中选择要求添加菜品的餐台号后添加，也可以在“开台列表”中选择要求添加菜品的餐台号，因为它与“台号”下拉列表框是联动的，即当在“台号”下拉列表框中选择餐台号后，如果在“开台列表”中存在该台号，对应的行也将被选中；如果更改“开台列表”中的选中行，在“台号”下拉列表框中也将更改为选中的餐台号。

## 16.8.2 开台签单工作区技术分析

在开发开台签单工作区时，为了使系统更人性化、智能化，要充分利用各种事件监听器。

例如，为“台号”下拉列表框添加 `ActionListener` 监听器，用来捕获下拉列表框选项被改变的事件，目的是同步处理“开台列表”和“签单列表”中的信息；为“商品（编号/助记码）”文本框添加 `KeyListener` 监听器，用来捕获在该文本框中按键被释放的事件，目的是跟踪用户输入内容同步获取最接近的商品，尽量让用户输入最少的内容就能得到需要的商品；为“数量”文本框添加 `FocusListener` 监听器，用来捕获该文本框获得或失去焦点的事件，因为默认数量为 1，当获得焦点时将数量设置为空，用户在改变数量时就不用先删除默认数量 1 了，当失去焦点时，查看用户是否输入了数量，如果未输入，则恢复默认数量 1；为“开台列表”表格添加 `MouseListener` 监听器，用来捕获表格行被选中的事件，目的是同步处理“签单列表”和“台号”下拉列表框中的信息。







### 16.8.3 开台签单工作区实现过程

■ 本模块使用的数据表: tb\_order\_item、tb\_menu、tb\_dest

首先,为“台号”下拉列表框添加事件监听器,用来处理开台或点菜的相关信息。如果选中的台号尚未开台(即新开台),则取消选择“开台列表”中的选中行,并清空“签单列表”中的所有行;如果选中的台号已经开台(即添加菜品),并且在“开台列表”中该台号未被选中,则选中“开台列表”中的该台号,并刷新“签单列表”中的菜品信息,即显示为当前选中台号所点的菜品。具体代码如下:

```
numComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int rowCount = rightTable.getRowCount(); // 获得开台列表中的行数,即已开台数
        if (rowCount > 0) { // 已经有开台
            // 获得“台号”下拉列表框中选中的台号
            String selectedDeskNum = numComboBox.getSelectedItem().toString();
            int needSelectedRow = -1; // 默认选中的台号未开台
            // 通过循环查看选中的台号是否已经开台
            opened: for (int row = 0; row < rowCount; row++) {
                // 获得已开台的台号
                String openedDeskNum = rightTable.getValueAt(row, 1).toString();
                // 查看选中的台号是否已经开台
                if (selectedDeskNum.equals(openedDeskNum)) {
                    needSelectedRow = row; // 选中的台号已经开台
                    break opened; // 跳出循环
                }
            }
            if (needSelectedRow == -1) { // 选中的台号尚未开台,即新开台
                rightTable.clearSelection(); // 取消选择“开台列表”中的选中行
                leftTableValueV.removeAllElements(); // 清空“签单列表”中的所有行
                // 刷新“签单列表”
                leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);
            } else { // 选中的台号已经开台,即添加菜品
                if (needSelectedRow != rightTable.getSelectedRow()) {
                    // “台号”下拉菜单中选中的台号在“开台列表”中未被选中
                    // 在“开台列表”中选中该台号
                    rightTable.setRowSelectionInterval(needSelectedRow);
                    // 清空“签单列表”中的所有行
                    leftTableValueV.removeAllElements();
                    // 将选中台号的签单列表添加到“签单列表”中
                    leftTableValueV.addAll(menuOfDeskV.get(needSelectedRow));
                    leftTableModel.setDataVector(leftTableValueV,
                        leftTableColumnV); // 刷新“签单列表”
                    // 选中“签单列表”中的第一行
                    leftTable.setRowSelectionInterval(0);
                }
            }
        }
    }
});
```

然后,开发智能获取点菜功能,通过为文本框添加键盘事件监听器实现。当按下的是【Enter】键时,等同于单击“开单”按钮,执行开单操作,在后面详细讲解具体操作过程;如果按下的不是【Enter】键,则获取输入的内容,同时判断输入的是商品编号,还是商品助记码,并按指定条件查询所有符合条件的菜品,如果存在符合条件的菜品,则获







取第一个符合条件的菜品，并显示菜品名称和单位，否则将菜品名称和单位设置为空。具体代码如下：

```
codeTextField.addKeyListener(new KeyAdapter() {
    public void keyReleased(KeyEvent e) { // 通过键盘监听器实现智能获取菜品
        if (e.getKeyCode() == KeyEvent.VK_ENTER) { // 按下【Enter】键
            makeOutAnInvoice(); // 开单
        } else {
            String input = codeTextField.getText().trim(); // 获得输入内容
            Vector vector = null; // 符合条件的菜品集合
            if (input.length() > 0) { // 输入内容不为空
                if (codeRadioButton.isSelected()) { // 按码查询
                    vector = dao.sMenuByCode(input); // 查询符合条件的菜品
                    if (vector.size() > 0) // 存在符合条件的菜品
                        vector = (Vector) vector.get(0); // 获得第一个符合条件的菜品
                } else {
                    vector = null; // 不存在符合条件的菜品
                } // 按编号查询
                vector = dao.sMenuById(input); // 查询符合条件的菜品
                if (vector.size() > 0) // 存在符合条件的菜品
                    // 获得第一个符合条件的菜品
                    vector = (Vector) vector.get(0);
                else
                    vector = null; // 不存在符合条件的菜品
            }
        }
        if (vector == null) { // 不存在符合条件的菜品
            nameTextField.setText(null); // 设置“商品名称”文本框为空
            unitTextField.setText(null); // 设置“单位”文本框为空
        } else { // 存在符合条件的菜品
            // 设置“商品名称”文本框为符合条件的菜品名称
            nameTextField.setText(vector.get(3).toString());
            // 设置“单位”文本框为符合条件的菜品单位
            unitTextField.setText(vector.get(5).toString());
        }
    }
});
```

下面的代码将实现一个智能化的用来填写数量的文本框，默认数量为 1，当文本框获得焦点时，自动将文本框设置为空；当文本框失去焦点时，将查看文本框是否输入了内容，如果未输入内容，则仍设置为默认数量 1。具体代码如下：

```
amountTextField.addFocusListener(new FocusListener() {
    public void focusGained(FocusEvent e) { // 当文本框获得焦点时执行
        amountTextField.setText(null); // 设置“数量”文本框为空
    }
    public void focusLost(FocusEvent e) { // 当文本框失去焦点时执行
        String amount = amountTextField.getText(); // 获得输入的数量
        if (amount.length() == 0) // 未输入数量
            amountTextField.setText("1"); // 恢复为默认数量 1
    }
});
amountTextField.setText("1"); // 默认数量为 1
```

如果在输入商品编号或助记码时按下【Enter】键，或者单击“开单”按钮，将执行开台点菜操作。如果是新开台点菜，则需要先处理开台信息，即在“开台列表”中添加新开台的信息，然后再处理点菜信息，即在“签单列表”中添加新点菜的信息；如果是为已







开台添加菜品, 则直接处理点菜信息。具体代码如下:

```
private void makeOutAnInvoice() {
    String deskNum = numComboBox.getSelectedItem().toString(); // 获得台号
    String menuName = nameTextField.getText(); // 获得商品名称
    String menuAmount = amountTextField.getText(); // 获得数量
    // 验证
    if (deskNum.equals("请选择")) { // 验证是否已经选择台号
        JOptionPane.showMessageDialog(null, "请选择台号!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    if (menuName.length() == 0) { // 验证是否已经确定商品
        JOptionPane.showMessageDialog(null, "请录入商品名称!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    // 验证数量是否有效, 数量必须在 1~99 之间
    if (!Validate.execute("[1-9][1][0-9]{0,1}", menuAmount)) {
        String info[] = new String[] { "您输入的数量错误!", "数量必须在 1~99 之间!" };
        JOptionPane.showMessageDialog(null, info, "友情提示",
            JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    // 处理开台信息
    int rightSelectedRow = rightTable.getSelectedRow(); // 获得被选中的台号
    int leftRowCount = 0; // 默认点菜数量为 0
    if (rightSelectedRow == -1) { // 没有被选中的台号, 即新开台
        rightSelectedRow = rightTable.getRowCount(); // 被选中的台号为新开的台
        Vector deskV = new Vector(); // 创建一个代表新开台的向量对象
        deskV.add(rightSelectedRow + 1); // 添加开台序号
        deskV.add(deskNum); // 添加开台号
        deskV.add(Today.getTime()); // 添加开台时间
        rightTableModel.addRow(deskV); // 将开台信息添加到“开台列表”中
        rightTable.setRowSelectionInterval(rightSelectedRow); // 选中新开的台
        menuOfDeskV.add(new Vector()); // 添加一个对应的签单列表
    } else { // 选中的台号已经开台, 即添加菜品
        leftRowCount = leftTable.getRowCount(); // 获得已点菜的数量
    }
    // 处理点菜信息
    Vector vector = dao.sMenuByNameAndState(menuName, "销售"); // 获得被点菜品
    int amount = Integer.valueOf(menuAmount); // 将菜品数量转为 int 型
    // 将菜品单价转为 int 型
    int unitPrice = Integer.valueOf(vector.get(5).toString());
    int money = unitPrice * amount; // 计算菜品消费额
    Vector<Object> menuV = new Vector<Object>();
    menuV.add("NEW"); // 添加新点菜标记
    menuV.add(leftRowCount + 1); // 添加点菜序号
    menuV.add(vector.get(0)); // 添加菜品编号
    menuV.add(menuName); // 添加菜品名称
    menuV.add(vector.get(4)); // 添加菜品单位
    menuV.add(amount); // 添加菜品数量
    menuV.add(unitPrice); // 添加菜品单价
    menuV.add(money); // 添加菜品消费额
    // 将点菜信息添加到“签单列表”中
    leftTableModel.addRow(menuV);
    leftTable.setRowSelectionInterval(leftRowCount); // 将新点菜设置为选中行
    // 将新点菜信息添加到对应的签单列表
    menuOfDeskV.get(rightSelectedRow).add(menuV);
    codeTextField.setText(null);
}
```







```
nameTextField.setText(null);
unitTextField.setText(null);
amountTextField.setText("1");
}
```

在新添加菜品的前方会有一个 NEW 标记，确定点菜结束后单击“签单”按钮，将取消所有新添加菜品前方的 NEW 标记。在未取消 NEW 标记的情况下可以选中后单击“取消”按钮取消该菜品，如果该餐台只点了该菜品，取消该菜品后将同时取消该餐台的开台信息；如果该餐台已经点了其他菜品，并且取消的不是最后点的菜品，还需要修改所点菜品的序号。具体代码如下：

```
// 获得选中菜品的新点菜标记
String NEW = leftTable.getValueAt(lSelectedRow, 0).toString();
if (NEW.equals("")) { // 没有新点菜标记，不允许取消
    JOptionPane.showMessageDialog(null, "很抱歉，该商品已经不能取消！", "友情提示",
        JOptionPane.INFORMATION_MESSAGE);
    return;
} else {
    // 获得“开台列表”中的选中行，即取消菜品的台号
    int rSelectedRow = rightTable.getSelectedRow();
    int i = JOptionPane.showConfirmDialog(null, "确定要取消" +
        rightTable.getValueAt(rSelectedRow, 1) + "中的商品" + leftTable.getValueAt(
            lSelectedRow, 3) + "吗?", "友情提示", JOptionPane.YES_NO_OPTION);
    // 弹出提示信息确认是否取消
    if (i == 0) { // 确认取消
        leftTableModel.removeRow(lSelectedRow); // 从“签单列表”中取消菜品
        int rowCount = leftTable.getRowCount(); // 获得取消后的点菜数量
        if (rowCount == 0) { // 未点任何菜品
            rightTableModel.removeRow(rSelectedRow); // 取消开台
            menuOfDeskV.remove(rSelectedRow); // 移除签单列表
        } else {
            if (lSelectedRow == rowCount) { // 取消菜品为最后一个
                lSelectedRow -= 1; // 设置最后一个菜品为选中的
            } else { // 取消菜品不是最后一个
                Vector<Vector<Object>> menus = menuOfDeskV.get(rSelectedRow);
                // 修改点菜序号
                for (int row = lSelectedRow; row < rowCount; row++) {
                    leftTable.setValueAt(row + 1, row, 1);
                    menus.get(row).set(1, row + 1);
                }
            }
            leftTable.setRowSelectionInterval(lSelectedRow); // 设置选中行
        }
    }
}
```

当用户要求添加菜品时，可以在“台号”下拉列表框中选择要求添加菜品的台号，也可以在“开台列表”中选择要求添加菜品的台号，在这里选择后将同步选中“台号”下拉列表框中的相应台号。具体代码如下：

```
rightTable.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        // 获得“开台列表”中的选中行
        int rSelectedRow = rightTable.getSelectedRow();
        leftTableValueV.removeAllElements(); // 清空“签单列表”中的所有行
        // 将选中台号的签单列表添加到“签单列表”中
        leftTableValueV.addAll(menuOfDeskV.get(rSelectedRow));
        // 刷新“签单列表”
    }
});
```







```

leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);
leftTable.setRowSelectionInterval(0); // 选中“签单列表”中的第一行
// 同步选中“台号”下拉菜单中的相应台号
numComboBox.setSelectedItem(rightTable.getValueAt(rSelectedRow, 1));
}
});

```

## 16.8.4 单元测试

在测试快速获取商品名称功能时,输入部分助记码后再输入一个空格,在“商品名称”文本框中仍然显示输入空格之前的商品名称,如图 16.25 所示。



图 16.25 输入空格后的效果

其中有两个小错误,一个错误是输入空格后就不应该显示原来的商品名称,这是因为在获取输入内容时去掉了首尾空格。具体代码如下:

```
String input = codeTextField.getText().trim(); // 获取输入内容
```

另一个错误是该文本框就应该不允许输入空格,解决了这个错误,上面的小错误也就自然解决了。但是建议将去掉首尾空格的代码去掉,这样对提升软件性能还是有好处的。这也是一个良好的编码习惯,应该尽量避免编写没有必要的代码。

如果想让文本框不允许输入空格,可以通过重载 `KeyListener` 类的 `keyTyped(KeyEvent e)` 方法实现,通过该方法的入口参数 `e` 的 `getKeyChar()` 方法可以得到此次输入的字符,如果为空格则通过 `consume()` 方法销毁此次事件。具体代码如下:

```

public void keyTyped(KeyEvent e) {
    if (e.getKeyChar() == ' ') // 判断用户输入的是否为空格
        e.consume(); // 如果是空格则销毁此次按键事件
}

```

同样,对“数量”文本框也可以采用这种办法控制用户输入的内容,并且可以控制输入数量的最大位数和不允许输入的第一位为 0。具体代码如下:

```

amountTextField.addKeyListener(new KeyAdapter() {
    public void keyTyped(KeyEvent e) {
        int length = amountTextField.getText().length(); // 获取当前数量的位数
        if (length < 2) { // 位数小于两位
            // 将允许输入的字符定义成字符串
            String num = (length == 0 ? "123456789" : "0123456789");
            // 查看按键字符是否包含在允许输入的字符中
            if (num.indexOf(e.getKeyChar()) < 0)
                e.consume(); // 如果不包含在允许输入的字符中则销毁此次按键事件
        } else {
            e.consume(); // 如果不小于数量允许的最大位数则销毁此次按键事件
        }
    }
});

```





## 16.9 自动结账工作区设计

### 16.9.1 自动结账工作区功能概述

自动结账工作区有两个主要功能，一个功能是自动计算当前选中餐台的消费金额，例如，选中餐台“8001”，在自动结账工作区将显示该餐台的消费金额，如图 16.26 所示。



图 16.26 选中的台号为“8001”

另一个功能是在结账时自动计算找零金额。用户输入“实收金额”后单击“结账”按钮，系统将自动计算出需要找零的金额，并弹出一个结账完成的提示框，如图 16.27 所示。

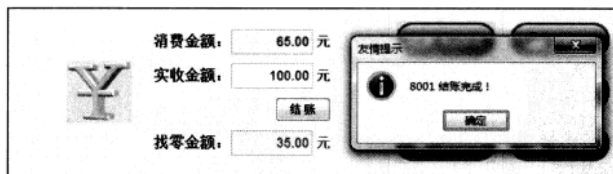


图 16.27 结账



### 16.9.2 自动结账工作区技术分析

如果要实现自动计算当前选中餐台消费金额的功能，就要随时监控“签单列表”中内容的变化。例如，添加或取消菜品，或者是“开台列表”中的选中行发生了改变，都将导





致“签单列表”中的内容发生改变。如果希望随时监控表格中内容的变化,可通过为表格模型添加 TableModelListener 监听器实现,无论是向表格模型中添加行,还是从表格模型中移除行,以及修改表格中某一单元格的值,都将触发该事件。

### 16.9.3 自动结账工作区实现过程

■ 本模块使用的数据表: tb\_order\_form、tb\_order\_item

(1) 实现自动计算当前选中餐台消费金额的功能,为与“签单列表”对应的表格模型添加一个 TableModelListener 监听器,在监听器中通过循环重新计算该餐台的消费金额,并更新“消费金额”文本框。具体代码如下:

```
leftTableModel.addTableModelListener(new TableModelListener() {
    // 通过表格模型监听器实现自动结账
    public void tableChanged(TableModelEvent e) {
        int rowCount = leftTable.getRowCount(); // 获得签单列表中的行数
        float expenditure = 0.0f; // 默认消费 0 元
        for (int row = 0; row < rowCount; row++) { // 通过循环计算消费金额
            expenditure += Float.valueOf(leftTable.getValueAt(row,
7).toString()); // 累加消费金额
        }
        expenditureTextField.setText(expenditure + "0");// 更新“消费金额”文本框
    }
});
```

(2) 实现结账功能。在结账之前首先要判断是否有未签单的菜品,如果有则弹出提示信息,如果没有则获得消费金额和实收金额;然后判断实收金额是否小于消费金额,如果小于消费金额,同样弹出提示,否则进行结账操作,将消费信息持久化到数据库,并弹出结账完成的提示框;最后将“实收金额”文本框和“找零金额”文本框设置为默认值。具体代码如下:

```
int rowCount = leftTable.getRowCount(); // 获得结账餐台的点菜数量
// 获得最后点菜的标记
String NEW = leftTable.getValueAt(rowCount - 1, 0).toString();
if (NEW.equals("NEW")) { // 如果最后点菜被标记为“NEW”,则弹出提示
    JOptionPane.showMessageDialog(null, "请先确定未签单商品的处理方式!", "友情提示",
JOptionPane.INFORMATION_MESSAGE);
} else {
    // 获得消费金额
    float expenditure = Float.valueOf(expenditureTextField.getText());
    // 获得实收金额
    float realWages = Float.valueOf(realWagesTextField.getText());
    if (realWages < expenditure) { // 如果实收金额小于消费金额,则弹出提示
        if (realWages == 0.0f)
            JOptionPane.showMessageDialog(null, "请输入实收金额!", "友情提示",
JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null, "实收金额不能小于消费金额!", "友情提示",
JOptionPane.INFORMATION_MESSAGE);
        realWagesTextField.requestFocus(); // 并令“实收金额”文本框获得焦点
    } else {
        // 计算并设置“找零金额”
        changeTextField.setText(realWages - expenditure + "0");
    }
}
```





```

        String[] values = { getNum(), rightTable.getValueAt(selectedRow,
1).toString(), Today.getDate() + " " + rightTable.getValueAt(selectedRow, 2),
expenditureTextField.getText(), user.get(0).toString() }; // 组织消费单信息
        dao.iOrderForm(values); // 持久化到数据库
        values[0] = dao.sOrderFormOfMaxId(); // 获得消费单编号
        for (int i = 0; i < rowCount; i++) { // 通过循环获得各个消费项目的信息
            values[1] = leftTable.getValueAt(i, 2).toString(); // 获得商品编号
            values[2] = leftTable.getValueAt(i, 5).toString(); // 获得商品数量
            // 获得商品消费金额
            values[3] = leftTable.getValueAt(i, 7).toString();
            dao.iOrderItem(values); // 持久化到数据库
        }
        JOptionPane.showMessageDialog(null, rightTable.getValueAt(selectedRow,
1) + " 结账完成!", "友情提示", JOptionPane.INFORMATION_MESSAGE); // 弹出结账完成提示
        rightTableModel.removeRow(selectedRow); // 从“开台列表”中取消开台
        leftTableValueV.removeAllElements(); // 清空“签单列表”
        // 刷新“签单列表”
        leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);
        realWagesTextField.setText("0.00"); // 清空“实收金额”文本框
        changeTextField.setText("0.00"); // 清空“找零金额”文本框
        menuOfDeskV.remove(selectedRow); // 从“签单列表”集合中移除已结账的签单列表
    }
}

```

## 16.10 结账报表工作区设计

### 16.10.1 结账报表工作区功能概述



图 16.28 结账报表工作区

本系统提供了 3 种方式的结账报表，分别是日结账报表、月结账报表和年结账报表，在结账报表工作区只提供了打开这 3 种结账报表功能的按钮，如图 16.28 所示。

日结账报表功能提供了对一日营业情况的统计，包括日开台数量、各个餐台的消费金额、菜品的消费情况、各个菜品的日销售情况，以及日营业额等，如图 16.29 所示。

编号	台号	开始时间	消费金额	红*狮子头	虾*馄饨	雪*大扁山	水*鸡片	肉*	可乐	水饺
20110216001	9001	16 16:16	65	---	---	---	---	---	---	---
20110216002	9001	16 16:45	65	---	---	---	---	---	---	---
总计	---	---	130	0	0	0	0	0	0	0

图 16.29 日结账报表

月结账报表功能提供了对一个月营业情况的统计，包括日开台总数、日总营业额、日





开台的平均消费额、日开台的最大和最小消费额, 当月的总开台数、月总营业额, 以及一个月中的日平均营业额、一个月中开台的最大和最小消费额, 如图 16.30 所示。

日期	开台总数	消费总额	平均消费额	最大消费额	最小消费额
1	--	--	--	--	--
2	--	--	--	--	--
3	--	--	--	--	--
4	--	--	--	--	--
5	--	--	--	--	--
6	--	--	--	--	--
7	--	--	--	--	--
8	--	--	--	--	--
9	--	--	--	--	--
10	--	--	--	--	--
11	--	--	--	--	--
12	--	--	--	--	--
13	--	--	--	--	--
14	--	--	--	--	--
15	--	--	--	--	--
16	2	130	65	65	65

图 16.30 月结账报表

年结账报表功能提供了对一年营业情况的统计, 包括一年中每天的营业额、每月的营业额、每月同一日期的总营业额, 以及一年的营业额, 如图 16.31 所示。

日期	1月	2月	3月	4月	5月	6月	7月	8月
1	--	--	--	--	--	--	--	--
2	--	--	--	--	--	--	--	--
3	--	--	--	--	--	--	--	--
4	--	--	--	--	--	--	--	--
5	--	--	--	--	--	--	--	--
6	--	--	--	--	--	--	--	--
7	--	--	--	--	--	--	--	--
8	--	--	--	--	--	--	--	--
9	--	--	--	--	--	--	--	--
10	--	--	--	--	--	--	--	--
11	--	--	--	--	--	--	--	--
12	--	--	--	--	--	--	--	--
13	--	--	--	--	--	--	--	--
14	--	--	--	--	--	--	--	--
15	--	--	--	--	--	--	--	--
16	--	130	--	--	--	--	--	--
17	--	--	--	--	--	--	--	--
18	--	--	--	--	--	--	--	--
19	--	--	--	--	--	--	--	--

图 16.31 年结账报表

## 16.10.2 结账报表工作区技术分析

在实现结账报表功能时, 有两个技术要点。

### 1. 对日期有效性的控制

例如, 在实现日结账功能时, 无论用户修改了统计日期的年度和月份, 都要影响到日下拉列表框中的可选项, 包括大月 (31 天) 和小月 (30 天) 的变化, 以及 2 月份在平年 (28 天) 和闰年 (29 天) 的变化, 如果不能正确处理这些变化, 将导致系统无法正常运行。其实, 在实现月结账报表和年结账报表时也会涉及这个问题, 只是在系统界面上不会





明显地体会到。解决该问题的大体思路是通过为年度和月份下拉列表框添加事件监听器，实现对日下拉列表框可选项的控制。

## 2. 对统计表格的控制

当系统界面不能显示出所有统计记录时，只需要将表格放到滚动面板中，这个办法对系统界面不能显示出统计记录的所有行很有效，因为在移动垂直滚动条时，表格的列名并不会随之滚动，即表格的列名是永远可见的；但是当系统界面不能显示出统计记录的所有列时，这个办法就不是很好了，因为在移动水平滚动条时，表格的所有列都会随之滚动，导致最左侧的一列或几列不可见，而表格最左侧的一列或者几列通常情况下也希望是永远可见的，即不会随着滚动条的移动而滚动。解决该问题的大体思路是实现两个表格，一个表格用来显示最左侧希望永远可见的一列或几列，另一个表格用来显示其他列，然后将两个表格并列显示。

## 16.10.3 结账报表工作区实现过程

### ■ 本模块使用的数据表：tb\_order\_form、tb\_order\_item

首先解决在实现结账报表功能时日期的有效性问题，需要定义一个数组，用来存放各个月份拥有的天数，默认2月份为28天。为了方便使用，将月份与数组的索引一一对应，即不使用数组索引为1的位置。具体代码如下：

```
private int daysOfMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

下面为年度下拉列表框添加事件监听器。首先获得选中的年度，并判断是平年还是闰年，以确定2月份的天数，如果为平年修改为28，为闰年则修改为29；然后获得当前选中的月份，如果当前选中的为2月份，则继续获得日下拉列表框拥有可选项的数量，当日下拉列表框拥有可选项的数量为28时，则为日下拉列表框添加一个可选项“29”；否则，从日下拉列表框中移除可选项“29”。具体代码如下：

```
yearComboBox.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int year = (Integer) yearComboBox.getSelectedItem(); // 获得选中的年度
        judgeLeapYear(year); // 判断是否为闰年，以确定2月份的天数
        // 获得选中的月份
        int month = (Integer) monthComboBox.getSelectedItem();
        if (month == 2) { // 如果选中的为2月
            // 获得日下拉列表框当前的天数
            int itemCount = dayComboBox.getItemCount();
            // 如果日下拉列表框当前的天数不等于2月份的天数
            if (itemCount != daysOfMonth[2]) {
                if (itemCount == 28) // 如果日下拉列表框当前的天数为28天
                    dayComboBox.addItem(29); // 则增加为29天
                else
                    // 否则日下拉列表框当前的天数为29天
                    dayComboBox.removeItem(29); // 则减少为28天
            }
        }
    }
});
```







下面为月份下拉列表框添加事件监听器。首先获得选中的月份,并获得日下拉列表框拥有可选项的数量。如果日下拉列表框拥有可选项的数量不等于当前选中月份拥有的天数,当日下拉列表框拥有可选项的数量大于当前选中月份拥有的天数时,则移除日下拉列表框中最大的可选项,并将日下拉列表框拥有可选项的数量减 1;否则将日下拉列表框拥有可选项的数量加 1,并添加到日下拉列表框的可选项中。具体代码如下:

```
monthComboBox.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获得选中的月份
        int month = (Integer) monthComboBox.getSelectedItem();
        int itemCount = dayComboBox.getItemCount(); // 获得日下拉列表框当前的天数
        // 如果日下拉列表框当前的天数不等于选中月份的天数
        while (itemCount != daysOfMonth[month]) {
            if (itemCount > daysOfMonth[month]) { // 如果大于选中月份的天数
                dayComboBox.removeItem(itemCount); // 则移除最后一个选择项
                itemCount--; // 并将日下拉列表框当前的天数减 1
            } else { // 否则小于选中月份的天数
                itemCount++; // 将日下拉列表框当前的天数加 1
                dayComboBox.addItem(itemCount); // 并添加为选择项
            }
        }
    }
});
```

通过年度和月份下拉列表框的事件监听器对日下拉列表框可选项的控制,无论选择哪一年或哪一月,日下拉列表框提供的日期可选项都是一个有效的日期。

下面解决当系统界面不能显示出统计记录的所有列时,移动滚动条导致最左侧的一列或几列不可见的问题。

(1) 通过实现抽象类 `AbstractTableModel`, 编写一个用来创建固定列表格模型的类 `FixedColumnTableModel`。具体代码如下:

```
class FixedColumnTableModel extends AbstractTableModel {
    @Override
    public int getColumnCount() { // 返回固定列的数量
        return fixedColumn;
    }
    @Override
    public int getRowCount() { // 返回行数
        return tableValueV.size();
    }
    @Override
    // 返回指定单元格的值
    public Object getValueAt(int rowIndex, int columnIndex) {
        return tableValueV.get(rowIndex).get(columnIndex);
    }
    @Override
    public String getColumnName(int columnIndex) { // 返回指定列的名称
        return tableColumnV.get(columnIndex);
    }
}
```

(2) 通过实现抽象类 `AbstractTableModel`, 编写一个用来创建移动列表格模型的类 `FloatingColumnTableModel`。具体代码如下:

```
class FloatingColumnTableModel extends AbstractTableModel {
    @Override
```





```

    public int getColumnCount() {                // 返回可移动列的数量
        return tableColumnV.size() - fixedColumn; // 需要扣除固定列的数量
    }
    @Override
    public int getRowCount() {                    // 返回行数
        return tableValueV.size();
    }
    @Override
    // 返回指定单元格的值
    public Object getValueAt(int rowIndex, int columnIndex) {
        // 需要为列索引加上固定列的数量
        return tableValueV.get(rowIndex).get(columnIndex + fixedColumn);
    }
    @Override
    public String getColumnName(int columnIndex) { // 返回指定列的名称
        // 需要为列索引加上固定列的数量
        return tableColumnV.get(columnIndex + fixedColumn);
    }
}

```



在处理与表格列有关的信息时，均需要在表格总列数的基础上减去固定列的数量。

下面通过实现接口 `ListSelectionListener`，编写一个用来同步两个表格中的选中行的事件监听器类 `MListSelectionListener`，即当选中固定列表格中的某一行时，监听器会同时选中移动列表格中的对应行。同样，当选中移动列表格中的某一行时，监听器会同时选中固定列表格中的对应行。具体代码如下：

```

class MListSelectionListener implements ListSelectionListener {
    boolean isFixedColumnTable = true; // 默认由选中固定列表格中的行触发
    public MListSelectionListener(boolean isFixedColumnTable) {
        this.isFixedColumnTable = isFixedColumnTable;
    }
    @Override
    public void valueChanged(ListSelectionEvent e) {
        if (isFixedColumnTable) { // 由选中固定列表格中的行触发
            // 获得固定列表格中的选中行
            int selectedRow = fixedColumnTable.getSelectedRow();
            // 同时选中可移动列表格中的选中行
            floatingColumnTable.setRowSelectionInterval(selectedRow);
        } else { // 由选中可移动列表格中的行触发
            // 获得可移动列表格中的选中行
            int selectedRow = floatingColumnTable.getSelectedRow();
            // 同时选中固定列表格中的选中行
            fixedColumnTable.setRowSelectionInterval(selectedRow);
        }
    }
}

```



(3) 依次创建固定列表格和移动列表格，并通过这两个表格的选择模型对象，为两个表格添加事件监听器；再创建一个滚动面板对象，并将固定列表格的列头对象添加到滚动面板的左上方；最后创建一个视口对象，先将固定表格对象添加到视口对象中，并将视口的首选大小设置为固定列表格的首选大小，并依次将视口和移动列表格添加到滚动面板的标题视口和默认视口中。具体代码如下：

```

fixedColumnTableModel = new FixedColumnTableModel(); // 创建固定列表格模型对象

```





```

fixedColumnTable = new MTable(fixedColumnTableModel); // 创建固定列表格对象
// 获得选择模型对象
ListSelectionModel fixed = fixedColumnTable.getSelectionModel();
// 添加行被选中的事件监听器
fixed.addListSelectionListener(new MListSelectionListener(true));
// 创建可移动列表格模型对象
floatingColumnTableModel = new FloatingColumnTableModel();
// 创建可移动列表格对象
floatingColumnTable = new MTable(floatingColumnTableModel);
// 获得选择模型对象
ListSelectionModel floating = floatingColumnTable.getSelectionModel();
// 添加行被选中的事件监听器
floating.addListSelectionListener(new MListSelectionListener(false));
JScrollPane scrollPane = new JScrollPane(); // 创建一个滚动面板对象
// 将固定列表格头放到滚动面板的左上方
scrollPane.setCorner(ScrollPaneConstants.UPPER_LEFT_CORNER,
fixedColumnTable.getTableHeader());
JViewport viewport = new JViewport(); // 创建一个用来显示基础信息的视口对象
viewport.setView(fixedColumnTable); // 将固定列表格添加到视口中
// 设置视口的首选大小为固定列表格的首选大小
viewport.setPreferredSize(fixedColumnTable.getPreferredSize());
scrollPane.setRowHeaderView(viewport); // 将视口添加到滚动面板的标题视口中
// 将可移动表格添加到默认视口
scrollPane.setViewportView(floatingColumnTable);

```

#### 16.10.4 单元测试

在测试年结账报表功能时,当系统界面不能显示出统计记录的所有列时,移动滚动条导致最左侧的一列或几列不可见的问题的确解决了;但是当选中右侧移动列表格的多行时,在左侧固定列表格中只选中了一行,如图 16.32 所示。当选中右侧移动列表格的第 26~30 行时,在左侧固定列表格中只选中了第 26 行。

日期	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	合计
12											
13											
14											
15											
16			130								
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											
28											
29											
30											
31											
总计		130									

图 16.32 选中移动列表格的多行

同样,当选中左侧固定列表格的多行时,在右侧移动列表格中也只选中了一行,如图 16.33 所示。当选中左侧固定列表格的第 26~30 行时,在右侧移动列表格中只选中了第 26 行。



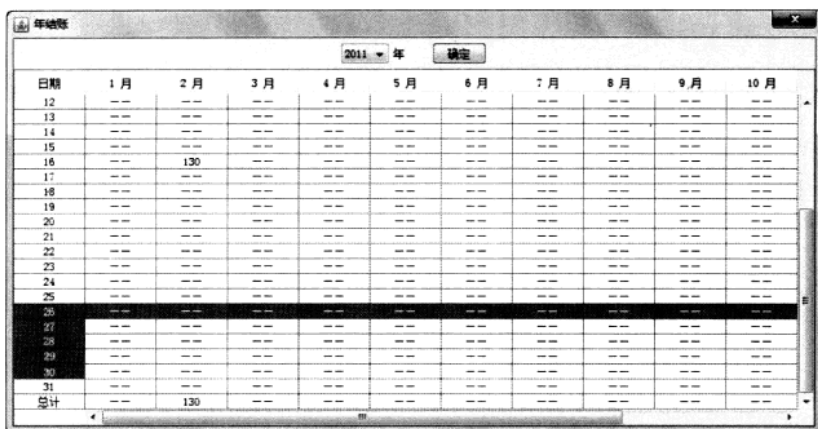


图 16.33 选中固定列表格的多行

如图 16.32 和图 16.33 所示的两种情况并不是想要的,这是因为在实现事件监听器时,并没有同步选中关联表中的所有对应行,而只是选中了关联表中的第一个对应行,但是表格却不是单选模式的,才导致出现这种情况。解决的办法是将表格设置为单选模式,因为这两个表格均是通过 FixedColumnTablePanel 类的内部类 MTable 实现的,所以,只需要在内部类 MTable 中重构其父类 JTable 的 getSelectionModel()方法。具体代码如下:

```
public ListSelectionModel getSelectionModel() {
    ListSelectionModel selectionModel = super.getSelectionModel();
    selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    return selectionModel;
}
```

## 16.11 后台管理工作区设计

后台管理工作区用来维护软件正常运行需要的一些信息,例如,台号信息、菜系信息、菜品信息,只有填写了这些信息,才能进行开台,以至结账和生成报表。

### 16.11.1 后台管理工作区功能概述

■ 本模块使用的数据表: tb\_menu、tb\_sort

在后台管理工作区提供了对台号、菜系和菜品信息的维护功能。在添加信息时,一是验证数据的合法性,例如,在添加台号信息时,不小心将座位数输入为了 100,在单击“添加”按钮时将弹出座位数输入错误的提示,如图 16.34 所示。再就是查看新添加的信息是否已经存在,例如,在添加菜系信息时,输入“炖炒类”类后单击“添加”按钮,将弹出菜系已经存在的提示,因为添加同名的菜系是没有意义的,如图 16.35 所示。





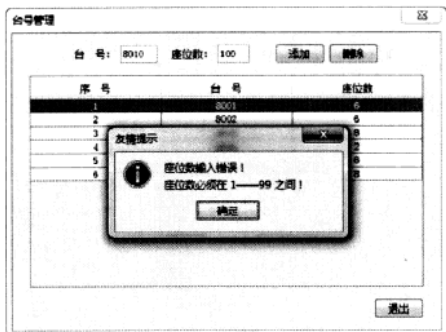


图 16.34 餐台座位数输入错误

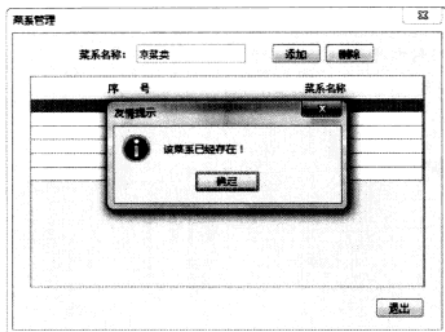


图 16.35 添加的菜品已经存在

在删除信息之前,通常情况下建议弹出一个确认提示框,以免由于疏忽错误删除信息,如图 16.36 所示。



图 16.36 删除菜品之前弹出的确认提示框

### 16.11.2 后台管理工作区技术分析

在对用户输入的数据进行验证时,如果某个数据不符合要求,通常情况下希望对应的控件获得焦点。如果是对用户输入的数据进行逐个验证,这个问题就很好地解决了;但是当对用户输入的数据进行批量验证时,就很难判断是哪个控件接收的数据不符合要求。这个问题可以通过 Java 的反射机制解决。通过 Java 反射可以轻松地对控件的内容进行批量验证,并且在接收数据不符合要求的情况下,可以直接令相应的控件获得焦点,另外,通过为控件设置名称,还可以弹出一个很有针对性的提示。

例如,在实现添加菜品功能时,就是通过 Java 的反射机制实现对 4 个文本框的不允许为空的验证,在未输入单价的情况下直接单击“添加”按钮,就会弹出一个很有针对性的提示,如图 16.37 所示。单击提示框中的“确定”按钮后,“单价”文本框将获得焦点,如图 16.38 所示。



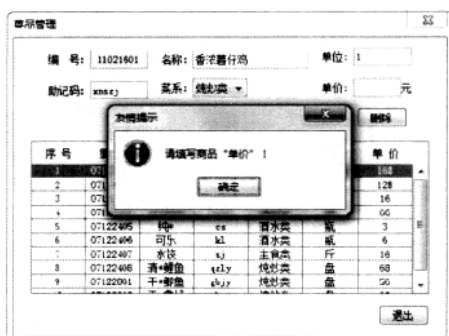


图 16.37 有针对性的提示内容



图 16.38 为空的“单价”文本框获得焦点

### 16.11.3 后台管理工作区实现过程

■ 本模块使用的数据表: tb\_menu、tb\_sort、tb\_user

后台管理包括对台号、菜系和菜品的管理,下面将依次讲解这3个功能的实现过程,以及一些典型的技巧。

#### 1. 实现台号管理功能

(1) 实现添加台号的功能。在执行添加台号操作时,首先要判断台号和座位数是否有效,台号最多为5个字符,座位数不能大于99个;然后创建一个向量对象,用来封装新添加台号的信息,并添加到表格中;最后将新添加的台号信息保存到数据库中。具体代码如下:

```
final JButton addButton = new JButton(); // 创建“添加台号”按钮对象
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String num = numTextField.getText().trim(); // 获取台号,并去掉首尾空格
        // 获取座位数,并去掉首尾空格
        String seating = seatingTextField.getText().trim();
        // 查看用户是否输入了台号和座位数
        if (num.equals("") || seating.equals("")) {
            JOptionPane.showMessageDialog(null, "请输入台号和座位数!", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        if (num.length() > 5) { // 查看台号的长度是否超过了5位
            JOptionPane.showMessageDialog(null, "台号最多只能为5个字符!", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            numTextField.requestFocus(); // 为台号文本框请求获得焦点
            return;
        }
        // 验证座位数是否在1~99之间
        if (!Validate.execute("[1-9]{1}([0-9]{0,1})", seating)) {
            String[] infos = { "座位数输入错误!", "座位数必须在1~99之间!" };
            JOptionPane.showMessageDialog(null, infos, "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            seatingTextField.requestFocus(); // 为“座位数”文本框请求获得焦点
        }
    }
});
```







```

        return;
    }
    if (dao.sDeskByNum(num) != null) { // 查看该台号是否已经存在
        JOptionPane.showMessageDialog(null, "该台号已经存在!", "友情提示",
            JOptionPane.INFORMATION_MESSAGE);
        numTextField.requestFocus(); // 为“台号”文本框请求获得焦点
        return;
    }
    int row = table.getRowCount(); // 获得当前拥有台号的个数
    Vector newDeskNumV = new Vector(); // 创建一个代表新台号的向量
    newDeskNumV.add(new Integer(row + 1)); // 添加添加序号
    newDeskNumV.add(num); // 添加台号
    newDeskNumV.add(seating); // 添加座位数
    tableModel.addRow(newDeskNumV); // 将新台号信息添加到表格中
    table.setRowSelectionInterval(row, row); // 设置新添加的台号为选中的
    numTextField.setText(null); // 将“台号”文本框设置为空
    seatingTextField.setText(null); // 将“座位数”文本框设置为空
    dao.iDesk(num, seating); // 将新添加的台号信息保存到数据库中
    JDBC.closeConnection(); // 关闭数据库连接
}
});
addButton.setText("添加");

```

(2) 实现删除台号的功能。在执行删除台号操作之前,首先要判断是否选中了要删除的台号;然后弹出提示确认是否真的要删除,如果真的要删除,还要判断该餐台是否正在被使用;最后执行删除操作。具体代码如下:

```

final JButton delButton = new JButton(); // 创建“删除台号”按钮对象
delButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int selectedRow = table.getSelectedRow(); // 获得选中的餐台
        if (selectedRow == -1) { // 未选中任何餐台
            JOptionPane.showMessageDialog(null, "请选择要删除的餐台!", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
        } else {
            // 获得选中餐台的编号
            String deskNum = table.getValueAt(selectedRow, 1).toString();
            // 查看该餐台是否正在被使用
            for (int row = 0; row < openedDeskTable.getRowCount(); row++) {
                if (deskNum.equals(openedDeskTable.getValueAt(row, 1))) {
                    JOptionPane.showMessageDialog(null, "该餐台正在使用,不能删除!",
                        "友情提示", JOptionPane.INFORMATION_MESSAGE);
                    return; // 该餐台正在被使用,不能删除,返回
                }
            }
            String infos[] = new String[] { // 组织确认信息
                "确定要删除餐台: ", "台号: " + deskNum, "座位数: " +
                table.getValueAt(selectedRow,
                    2).toString() };
            // 弹出确认提示
            int i = JOptionPane.showConfirmDialog(null, infos, "友情提示",
                JOptionPane.YES_NO_OPTION);
            if (i == 0) { // 确认删除
                dao.dDeskByNum(deskNum); // 从数据库中删除
                // 刷新表格
                tableModel.setDataVector(dao.sDesk(), columnNameV);
                int rowCount = table.getRowCount(); // 获得删除后拥有的餐台数
                if (rowCount > 0) { // 还拥有餐台
                    if (selectedRow == rowCount) // 删除的为最后一个餐台

```





```

        selectedRow -= 1; // 将选中的餐台前移一行
        // 设置当前选中的餐台
        table.setRowSelectionInterval(selectedRow, selectedRow);
    }
    JDBC.closeConnection(); // 关闭数据库连接
}
}
});
delButton.setText("删除");

```

## 2. 实现菜系管理功能

(1) 实现添加菜系的功能。在执行添加菜系操作时，首先要判断菜系名称的长度是否超出了允许的最大长度，并查看该菜系名称是否已经存在；然后创建一个向量对象，用来封装新添加菜系的信息，并添加到表格中；最后将新添加的菜系信息保存到数据库中。具体代码如下：

```

final JButton addButton = new JButton(); // 创建“添加菜系名称”按钮对象
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 获得菜系名称，并去掉首尾空格
        String sortName = sortNameTextField.getText().trim();
        if (sortName.equals("")) { // 查看是否输入了菜系名称
            JOptionPane.showMessageDialog(null, "请输入菜系名称！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        if (sortName.length() > 10) { // 查看菜系名称的长度是否超过了 10 个汉字
            JOptionPane.showMessageDialog(null, "菜系名称最多只能为 10 个汉字！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        if (dao.sSortByName(sortName).size() > 0) { // 查看该菜系名称是否已经存在
            JOptionPane.showMessageDialog(null, "该菜系已经存在！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        int row = tableModel.getRowCount(); // 获得当前拥有菜系名称的个数
        Vector newSortV = new Vector(); // 创建一个代表新菜系名称的向量
        newSortV.add(new Integer(row + 1)); // 添加序号
        newSortV.add(sortName); // 添加菜系名称
        tableModel.addRow(newSortV); // 将新菜系名称信息添加到表格中
        table.setRowSelectionInterval(row, row); // 设置新添加的菜系名称为选中的
        sortNameTextField.setText(null); // 将“菜系名称”文本框设置为空
        //
        dao.iSort(sortName); // 将新添加的菜系名称信息保存到数据库中
        JDBC.closeConnection(); // 关闭数据库连接
    }
});
addButton.setText("添加");

```



(2) 实现删除菜系的功能。在执行删除菜系操作之前，首先要判断是否有菜系被选中；然后弹出提示确认是否真的删除；最后执行删除操作，如果删除的是表格中的最后一行，则选中删除后表格中的最后一行，如果删除的不是最后一行，则选中删除后同一位置的表格行。具体代码如下：

```

final JButton delButton = new JButton(); // 创建删除菜系名称按钮对象

```





```

delButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int row = table.getSelectedRow();           // 获得选中的菜系
        // 获得选中的菜系名称
        String delSortName = (String) table.getValueAt(row, 1);
        int j = JOptionPane.showConfirmDialog(null, "确定要删除菜系" + delSortName
+ "? ", "友情提示", JOptionPane.YES_NO_OPTION);    // 弹出确认提示
        if (j == 0) {                               // 确认删除
            tableModel.removeRow(row);              // 从表格中移除菜系信息
            int rowCount = table.getRowCount();      // 获得删除后拥有的菜系数
            if (rowCount > 0) {                      // 还拥有菜系
                if (row < table.getRowCount()) { // 删除的不是位于表格最后的菜系
                    for (int i = row; i < table.getRowCount(); i++) {
                        // 修改位于删除菜系之后的序号
                        table.setValueAt(i + 1 + "", i, 0);
                    }
                    // 设置上移到删除行索引的菜系为被选中
                    table.setRowSelectionInterval(row, row);
                } else {                             // 删除的是位于表格最后的菜系
                    // 设置当前位于表格最后的菜系被选中
                    table.setRowSelectionInterval(row - 1, row - 1);
                }
            }
            dao.dSortByName(delSortName);           // 从数据库中删除菜系
            JDBC.closeConnection();                // 关闭数据库连接
        }
    }
});
delButton.setText("删除");

```

### 3. 实现菜品管理功能

(1) 实现添加菜品的功能。在执行添加菜品操作时, 首先通过 Java 反射机制验证 4 个文本框是否为空, 如果为空则弹出要求填写信息的提示框, 并且通过获取控件的标识名称, 组织出有针对性的提示信息, 还要为空的文本框请求获得焦点, 之后再对这些信息进行具体的验证; 然后创建一个向量对象, 用来封装新添加菜品的信息, 并添加到表格中; 最后将新添加的菜品信息保存到数据库中。具体代码如下:

```

final JButton addButton = new JButton();
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // 通过 Java 反射获取 MenuDialog 类的所有属性
        Field[] fields = MenuDialog.class.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            Field field = fields[i];                // 获得指定属性
            // 只验证 JTextField 类型的属性
            if (field.getType().equals(JTextField.class)) {
                field.setAccessible(true);          // 私有属性必须设为 true 才允许访问
                JTextField textField = null;         // 声明一个 JTextField 类型的对象
                try {
                    // 获得本类中的相应对象
                    textField = (JTextField) field.get(MenuDialog.this);
                } catch (Exception exception) {
                    exception.printStackTrace();
                }
                if (textField.getText().trim().equals("")) { // 文本框为空

```







```
JOptionPane.showMessageDialog(null, "请填写商品" +
    textField.getName() + "!",
    // 弹出需要输入信息的提示
    "友情提示", JOptionPane.INFORMATION_MESSAGE);
textField.requestFocus(); // 令文本框获得焦点
return; // 返回
}
}
if (sortComboBox.getSelectedIndex() == 0) { // 单独验证下拉列表框
    JOptionPane.showMessageDialog(null, "请选择商品所属“菜系”!", "友情提示",
        JOptionPane.INFORMATION_MESSAGE);
    return;
}
String menu[] = new String[7]; // 创建一个数组, 用来保存菜品信息
menu[0] = numTextField.getText().trim(); // 获得菜品编号
menu[1] = nameTextField.getText().trim(); // 获得菜品名称
menu[2] = codeTextField.getText().trim(); // 获得菜品助记码
menu[3] = sortComboBox.getSelectedItem().toString(); // 获得菜品所属菜系
menu[4] = unitTextField.getText().trim(); // 获得菜品单位
menu[5] = unitPriceTextField.getText().trim(); // 获得菜品单价
menu[6] = "销售";
if (menu[1].length() > 10) {
    JOptionPane.showMessageDialog(null, "菜品名称最多只能为 10 个汉字!", "友
友情提示", JOptionPane.INFORMATION_MESSAGE);
    nameTextField.requestFocus();
    return;
}
if (menu[2].length() > 10) {
    JOptionPane.showMessageDialog(null, "助记码最多只能为 10 个字符!", "友
情提示", JOptionPane.INFORMATION_MESSAGE);
    codeTextField.requestFocus();
    return;
}
if (menu[4].length() > 2) {
    JOptionPane.showMessageDialog(null, "单位最多只能为 2 个汉字!", "友情
提示", JOptionPane.INFORMATION_MESSAGE);
    unitTextField.requestFocus();
    return;
}
if (!Validate.execute("[1-9]{1}[0-9]{0,3}", menu[5])) {
    String infos[] = { "单价输入错误!", "单价必须在 1~9999" };
    JOptionPane.showMessageDialog(null, infos, "友情提示",
        JOptionPane.INFORMATION_MESSAGE);
    unitPriceTextField.requestFocus();
    return;
}
if (dao.sMenuByNameAndState(menu[1], "销售") != null) {
    JOptionPane.showMessageDialog(null, "该菜品已经存在!", "友情提示",
        JOptionPane.INFORMATION_MESSAGE);
    nameTextField.requestFocus();
    return;
}
int row = tableModel.getRowCount(); // 获得当前拥有的菜品数量
Vector newMenuV = new Vector();
newMenuV.add(row + 1); // 添加序号
for (int i = 0; i < menu.length; i++) {
```







```

        newMenuV.add(menu[i]); // 添加菜品信息
    }
    // 获得所属菜系
    Vector sortVector = (Vector) dao.sSortByName(menu[3]).get(0);
    menu[3] = sortVector.get(1).toString(); // 设置菜系主键
    Vector homonymyMenuOfDel = dao.sMenuByNameAndState(menu[1], "删除");
    if (homonymyMenuOfDel == null) {
        dao.iMenu(menu); // 将新菜品信息保存到数据库
    } else {
        newMenuV.set(1, homonymyMenuOfDel.get(0));
        dao.uMenu(menu);
    }
    tableModel.addRow(newMenuV); // 将新菜品添加到表格中
    table.setRowSelectionInterval(row, row); // 选中新添加的菜品
    numTextField.setText(getNextNum(menu[0]));
    nameTextField.setText(null);
    codeTextField.setText(null);
    sortComboBox.setSelectedIndex(0);
    unitTextField.setText(null);
    unitPriceTextField.setText(null);
}
});
addButton.setText("添加");

```

(2) 实现删除菜品的功能。在执行删除菜品操作之前, 首先要判断是否存在被选中的菜品; 然后弹出提示确认是否真的删除; 最后, 如果删除的不是表格的最后一行, 还要修改其后未删除菜品的序号。具体代码如下:

```

final JButton delButton = new JButton();
delButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int row = table.getSelectedRow(); // 获得选中的菜品
        String delMenuName = table.getValueAt(row, 2).toString();
        String info = "确定要删除菜品" + delMenuName + "吗? ";
        // 弹出确认提示框
        int j = JOptionPane.showConfirmDialog(null, info, "友情提示",
        JOptionPane.YES_NO_OPTION);
        if (j == 0) {
            // 确认删除
            tableModel.removeRow(row); // 从表格中移除菜品信息
            int rowCount = table.getRowCount(); // 获得删除后拥有的菜品数
            if (rowCount > 0) { // 还拥有菜品
                if (row < table.getRowCount()) { // 删除的不是位于表格最后的菜系
                    for (int i = row; i < table.getRowCount(); i++) {
                        // 修改位于删除菜系之后的序号
                        table.setValueAt(i + 1 + "", i, 0);
                    }
                    // 设置上移到删除行索引的菜系为被选中
                    table.setRowSelectionInterval(row, row);
                } else {
                    // 设置当前位于表格最后的菜系被选中
                    table.setRowSelectionInterval(row - 1, row - 1);
                }
            }
            dao.uMenuStateByName(delMenuName, "删除");
            JDBC.closeConnection();
        }
    }
});

```





```
});
delButton.setText("删除");
```

### 16.11.4 单元测试

当测试菜品管理功能时,在不填写任何菜品信息的情况下直接单击“添加”按钮,弹出了如图 16.39 所示的提示信息。

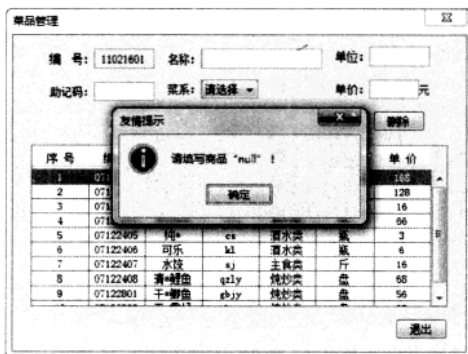


图 16.39 通过 Java 反射验证是否为空弹出的错误提示

这是因为没有为文本框控件设置标识名称,默认情况下文本框控件的标识名称为空,所以在通过文本框控件的 `getName()` 方法获得文本框控件的标识名称时才是“null”,解决的办法是通过 `setName(String name)` 方法为文本框控件设置标识名称。具体代码如下:

```
nameTextField = new JTextField();           //创建“名称”文本框对象
nameTextField.setName("名称");              //为“名称”文本框设置标识名称
codeTextField = new JTextField();           //创建“助记码”文本框对象
codeTextField.setName("助记码");           //为“助记码”文本框设置标识名称
unitTextField = new JTextField();           //创建“单位”文本框对象
unitTextField.setName("单位");             //为“单位”文本框设置标识名称
unitPriceTextField = new JTextField();      //创建“单价”文本框对象
unitPriceTextField.setName("单价");        //为“单价”文本框设置标识名称
```

## 16.12 开发问题解析

作为一名软件开发人员,在设计开发应用软件的过程中,要时刻从软件使用者的角度出发,力求开发出一款功能实用、界面简单、操作人性化并且智能化的产品,只有这样的产品,才更容易被用户接受。

笔者在开发本系统的过程中,就是严格按照这些要求实施的,主要有以下几点。

### 1. 通过输入少量内容就可以快速获取相关产品

通过 `KeyListener` 监听器,可以很方便地捕获各种键盘事件。`KeyListener` 监听器通过 3 个接口方法捕获 3 种不同类型的键盘事件:方法 `keyPressed(KeyEvent e)` 用来捕获键盘中的某个按键被按下的事件,当某个按键被按下时,将执行该方法;方法 `keyReleased(KeyEvent e)` 用来捕获键盘中的某个按键被释放的事件,当某个按键被释放时,



500







将执行该方法；方法 `keyTyped(KeyEvent e)` 用来捕获键入键盘中的某个键的事件，当键入某个键时，将执行该方法。

“按下键”和“释放键”事件是低级别事件，它们依赖于平台和键盘布局。只要按下或释放按键就会生成这些事件，这些事件是获取不生成字符输入的键（如动作键、组合键等）的唯一方式。通过 `KeyEvent` 类的 `getKeyCode()` 方法可以获得代表按下或释放键的虚拟键码，虚拟键码用于报告按下了键盘上的哪个键，而不是通过一个或多个击键组合所生成的字符（如 A 是由【Shift+A】生成的）。

“键入键”事件是高级别事件，通常情况下不依赖于平台或键盘布局。在输入 Unicode 字符时会生成此类事件，不生成 Unicode 字符的键是不会生成键入键事件的（如动作键、组合键等）。最简单的情况是按下单个键（如【A】）将产生键入键事件，但是经常是通过一系列按键（如【Shift+A】）来产生字符，并且按下键事件和键入键事件的映射关系可能是多对一或多对多的。键释放时通常情况下不需要生成一个键入键事件，但是在某些情况下，只有释放某个键才会生成键入键事件（如在 Windows 中通过 Alt-Numpad 方法来输入 ASCII 序列）。

利用控件的 `addKeyListener(KeyListener keyListener)` 方法可以将该监听器对象注册到控件中，这样，在按下、释放或键入键生成键盘事件时，将调用监听器对象中的相关方法，并传递过来一个 `KeyEvent` 对象。

## 2. 人性化控制商品数量 `focusLost(FocusEvent e)`

通过 `FocusListener` 监听器，可以很方便地捕获关于焦点的事件。`FocusListener` 监听器通过两个接口方法捕获两种不同类型的焦点事件：方法 `focusGained(FocusEvent e)` 用来捕获控件获得焦点的事件，当其监听的控件获得焦点时，将执行该方法；方法 `focusLost(FocusEvent e)` 用来捕获控件失去焦点的事件，当其监听的控件失去焦点时，将执行该方法。

焦点事件分为两个级别，分别是持久性的和暂时性的。如果焦点直接从一个控件移动到另一个控件，如通过调用 `requestFocus()` 方法，或者用户通过【Tab】键遍历控件时，则为持久性焦点更改事件。如果是由于另一个操作间接引起的控件暂时失去焦点，如释放窗口或拖动滚动条，则为暂时性焦点更改事件。在这种情况下，一旦该操作结束，将自动恢复到原始焦点状态。对于释放窗口的情况，只要重新激活窗口就能恢复到原始焦点状态。持久性焦点事件和暂时性焦点事件通过 `FOCUS_GAINED` 和 `FOCUS_LOST` 区分，可以通过 `isTemporary()` 方法判断事件的级别。

利用控件的 `addFocusListener(FocusListener focusListener)` 方法可以将该监听器对象注册到控件中，这样在控件获得或失去焦点时，将调用监听器对象中的相关方法，并传递过来一个 `FocusEvent` 对象。

## 3. 系统自动结账

通过 `TableModelListener` 监听器，可以很方便地捕获由表格模型发生变化产生的事件，包括向表格模型中添加行、从表格模型中移除行，以及修改某一单元格的值。`TableModelListener` 监听器只提供了一个接口方法 `tableChanged(TableModelEvent e)`。通过 `TableModelEvent` 对象可以判断触发此次事件的具体原因，例如，通过 `getType()` 方法的返回值判断是由向表格模型中添加行触发的，还是由从表格模型中移除行或修改某一单元格的值触发的，当返回值为静态常量 `INSERT` 时，说明是由向表格模型中添加行触发的。当





返回值为静态常量 `UPDATE` 时，说明是由从表格模型中移除行触发的；当返回值为静态常量 `DELETE` 时，则是由修改某一单元格的值触发的。

利用控件的 `addTableModelListener (TableModelListener tableModelListener)` 方法可以将该监听器对象注册到控件中，这样，在表格模型发生变化时，将调用监听器对象中的 `tableChanged(TableModelEvent e)` 方法，并传递过来一个 `TableModelEvent` 对象。





# 第 17 章

---

## 企业人事管理系统

( Swing+Hibernate+Oracle 实现 )

企业的发展不仅需要技术的竞争，市场的竞争，服务的竞争，还需要人才的竞争，并且成为市场竞争中一个重要的环节。优秀人才的引入将给企业的发展注入新鲜的血液，带给企业巨大的发展空间。所以，吸引人才、留住人才就成为了企业人事管理的一个重要课题。要想留住人才不仅需要企业具有良好的发展前景，更重要的是企业要有一个健全的管理体制，这不仅能节省企业大量的人力和物力，还可以提高企业的经济效益，从而带动企业快速发展。通过阅读本章，读者可以学习到：

- » 企业人事管理系统的软件结构和业务流程
- » 如何利用 Hibernate 建立持久层
- » Oracle 数据库的使用方法
- » Swing 中表格行选取事件的使用方法
- » Swing 中树状结构的使用和维护，例如，维护公司结构树
- » Swing 中选取并显示图片的方法
- » 通过 Java 反射验证数据是否为空
- » 页面中组件联动的实现方法
- » 如何开发一些简单、实用的功能模块，例如，支持树状结构的下拉菜单
- » 如何在程序中调用其他工具软件，例如，计算器、Excel 等



## 17.1 开发背景

飞速发展的技术变革和创新，以及迅速变化的差异化顾客需求等新竞争环境的出现，使得越来越多的组织通过构筑自身的人事竞争力来维持生存并促进持续发展。在“以人为本”观念的熏陶下，企业人事管理在组织中的作用日益突出。但是，人员的复杂性和组织的特有性使得企业人事管理成为难题。基于这个时代背景，企业人事管理便成为企业管理的重要内容。企业人事管理系统的作用之一是为企业的员工建立人事档案，它的出现使得人事档案查询、调用的速度加快，也使得精确分析大量员工的知识、经验、技术、能力和职业抱负成为可能，从而实现企业人事管理的标准化、科学化、数字化。

## 17.2 系统分析

伴随着企业人事管理系统化的日益完善，企业人事管理系统在企业管理中越来越受到企业管理者的青睐。企业人事管理系统的功能全面、操作简单，可以快速地为员工建立电子档案，并且便于修改、保存和查看，实现了无纸化存档，为企业节省了大量资金和空间。通过企业人事管理系统，还可以实现对企业员工的考勤管理、奖惩管理、培训管理、待遇管理和快速生成待遇报表。

## 17.3 系统设计

### 17.3.1 系统目标

根据企业对人事管理的要求，本系统需要实现以下目标。

- ☐ 操作简单方便、界面简洁大方。
- ☐ 方便、快捷的档案管理。
- ☐ 简单实用的考勤和奖惩管理。
- ☐ 简单实用的培训管理。
- ☐ 针对企业中不同的待遇标准，实现待遇账套管理。
- ☐ 简单明了的账套维护功能。
- ☐ 方便、快捷的账套人员设置。
- ☐ 功能强大的待遇报表功能。
- ☐ 系统运行稳定、安全可靠。



### 17.3.2 系统功能结构

企业人事管理系统主要包括人事管理和待遇管理两大功能模块，用来提供对企业员工



的人事和待遇管理。系统的辅助功能模块包括系统维护和用户管理,用来提供对系统的维护和系统安全;还包含一个系统工具模块,用来快速运行系统中的常用工具,例如,系统计算器和 Excel 表格等。

人事管理模块包含的子模块有档案管理、考勤管理、奖惩管理和培训管理。其中,档案管理模块用来维护员工的基本信息,包括档案信息、职务信息和个人信息。其中,档案信息包括员工的照片。档案信息只可以添加和修改,不可以删除,因为员工档案将作为企业的永久资源和历史记录进行保存。在维护员工档案时,可以通过企业结构树快速查找员工。考勤管理模块用来记录员工的考勤信息,例如迟到、请假、加班等。奖惩管理模块用来记录员工的奖惩信息,例如因为某事情奖励或惩罚员工。培训管理模块用来记录对员工的培训信息。

待遇管理模块包含的子模块有账套管理、人员设置和统计报表。其中,账套管理模块用来建立和维护账套。所谓账套,就是对不同员工采用不同的待遇标准。例如,对已经签定劳动合同的员工和处于试用期的员工的基本工资是不同的,针对这种情况可以分别建立一个试用期账套和合同工账套。这里假设处于试用期的员工的基本工资为 2000,而已经签定劳动合同的员工的基本工资为 3000,则可以分别将试用期账套和合同工账套中的基本工资项设为 2000 和 3000。账套中的部分项目可以用于考勤管理模块的考勤项目。人员设置模块用来设置对员工采用前面建立的哪个账套,即采用哪个待遇标准,如果没有适合的账套,则可以继续建立新的账套。统计报表模块将以表格的形式统计员工的待遇情况,这里将用到在考勤管理和奖惩管理模块填写的数据,可以按月、季度、半年和年统计。

系统维护模块包含的子模块有企业架构、基本资料和初始化系统。其中,企业架构模块用来维护企业的组织结构,企业架构将以树状结构显示;基本资料模块用来维护职务种类、用工形式、账套项目、考勤项目、民族和籍贯信息;初始化系统模块用来对系统进行初始化,在正式使用前需要对系统进行初始化。

用户管理模块包含的子模块有新增用户和修改密码。其中,新增用户模块用来添加和维护系统的管理员,包括冻结和删除管理员,该模块只有超级管理员有权使用;修改密码模块用来为当前登录用户修改登录密码。

系统工具模块包含的功能有打开计算器、打开 Word 和打开 Excel,以方便用户快速地打开这 3 个常用的系统工具。

企业人事管理系统的功能结构如图 17.1 所示。

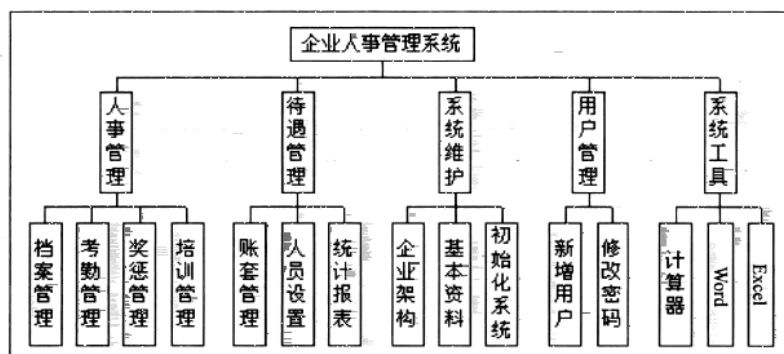


图 17.1 企业人事管理系统功能结构



### 17.3.3 系统预览

企业人事管理系统由多个界面组成，下面仅列出几个典型界面，其他界面效果可参见光盘中的源程序。

企业人事管理系统的主窗体效果如图 17.2 所示，窗体的左侧为系统的功能结构导航，窗体的上方为系统常用功能的快捷按钮。

单击图 17.2 中的左侧导航栏中的“档案管理”节点，将打开如图 17.3 所示的档案列表界面。单击界面上方的“新建员工档案”按钮，可以建立新的员工档案；单击左侧的企业架构树中的部门节点，在右侧将显示相应部门的员工列表，首先选中其中的一行，然后单击界面上方的“修改员工档案”按钮，可以修改选中员工的档案。

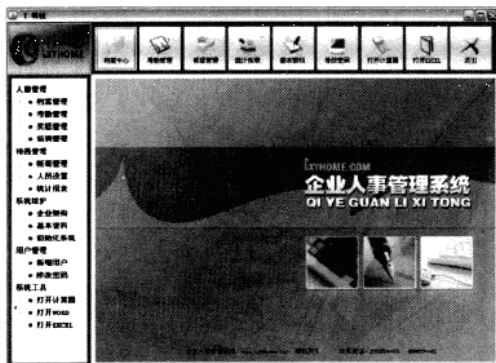


图 17.2 企业人事管理系统主窗体效果

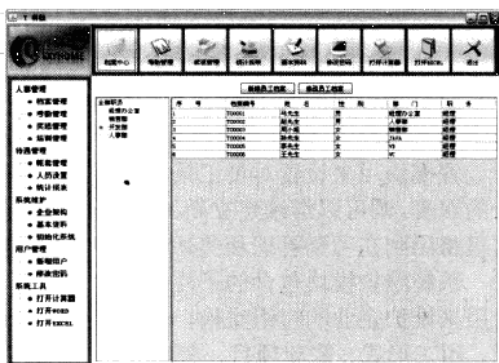


图 17.3 档案列表界面

单击图 17.2 中的左侧导航栏中的“培训管理”节点，将打开如图 17.4 所示的培训管理界面。在该界面可以建立培训信息，以及设置参训人员列表。

单击图 17.2 中的左侧导航栏中的“账套管理”节点，将打开如图 17.5 所示的账套管理界面。在该界面可以维护账套信息，主要包括建立账套、添加或删除账套项目，以及修改项目金额。

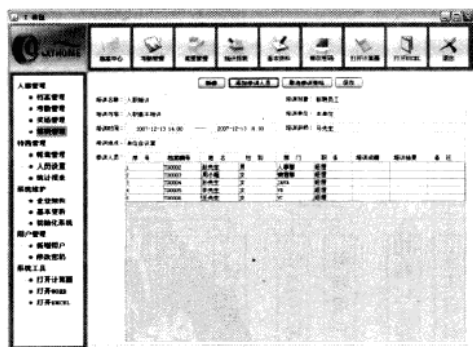


图 17.4 培训管理界面

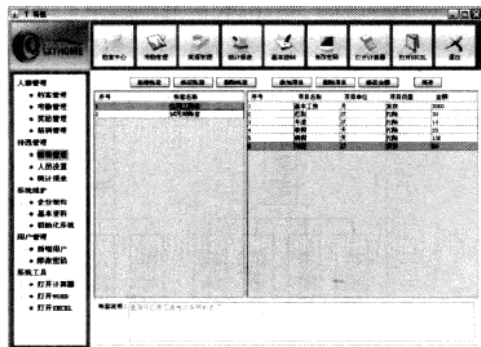


图 17.5 账套管理界面







单击图 17.2 中的左侧导航栏中的“统计报表”节点，将打开如图 17.6 所示的统计报表界面。在该界面可以生成统计报表，可以生成的报表种类有月报表、季度报表、半年报表和年度报表。

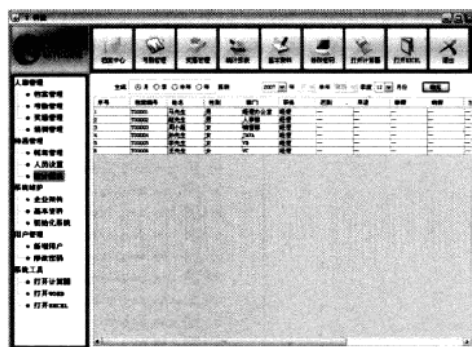


图 17.6 统计报表界面

### 17.3.4 业务流程图

企业人事管理系统的业务流程如图 17.7 所示。

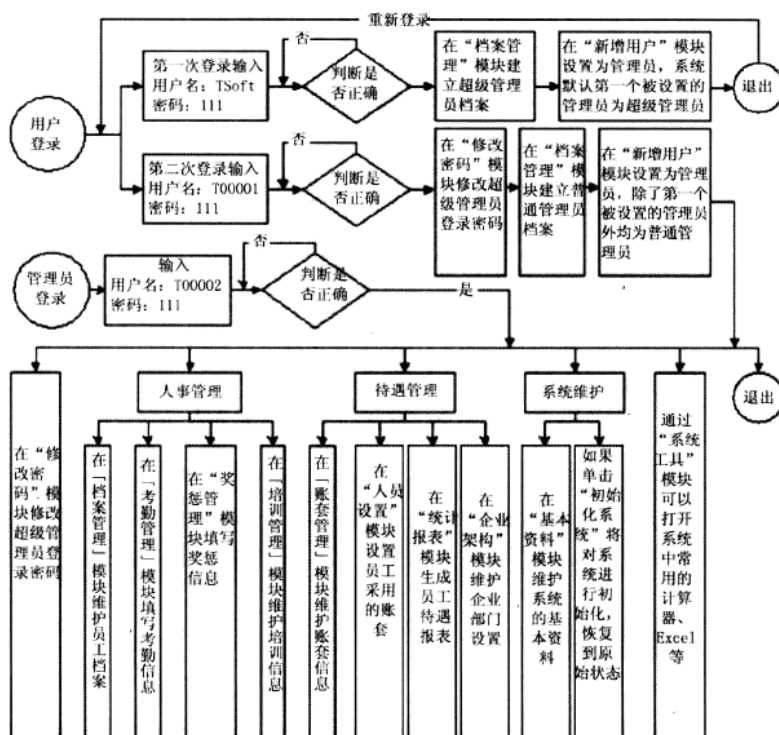


图 17.7 人事管理系统的业务流程图

### 17.3.5 文件夹结构设计

每个项目都会有相应的文件夹组织结构。当项目中的窗体过多时，为了便于查找和使





用,可以将窗体进行分类,放入不同的文件夹中,这样既便于前期开发,又便于后期维护。企业人事管理系统文件夹组织结构如图 17.8 所示。

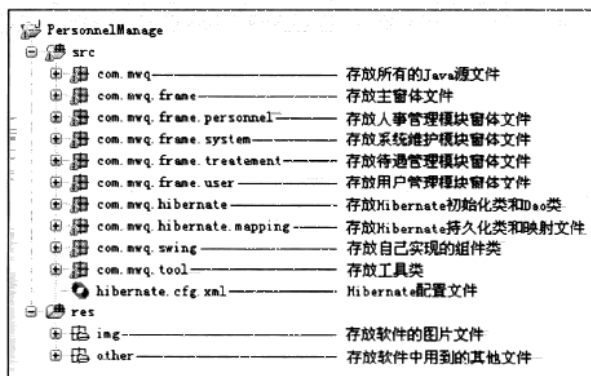


图 17.8 企业人事管理系统文件夹结构图

## 17.4 数据库设计

在开发应用程序时,对数据库的操作是必不可少的,而一个数据库的设计优秀与否,将直接影响到软件的开发进度和性能,所以,对数据库的设计就显得尤为重要。数据库的设计要根据程序的需求及其功能制定,如果在开发软件之前不能很好地设计数据库,在开发过程中将反复修改数据库,这将严重影响开发进度。

### 17.4.1 数据库分析

企业人事管理系统的需求主要包括对人事档案的管理,其中包括档案信息、职务信息和个人信息;人事考勤、奖惩、培训的管理,并且考勤和奖惩信息将体现到待遇统计当中;待遇管理,还要针对企业的现实需求,要求企业人事管理系统支持多账套功能。

### 17.4.2 数据库概念设计

数据库设计是系统设计过程中的重要组成部分,它是通过管理系统的整体需求而制定的,数据库设计的好坏直接影响到系统的后期开发。下面对本系统中具有代表性的数据库设计进行详细说明。

在开发企业人事管理系统时,最重要的是人事档案信息。本系统将档案信息分为档案信息、职务信息和个人信息,由于信息多而复杂,这里只给出关键的信息。档案信息表 E-R 图如图 17.9 所示。





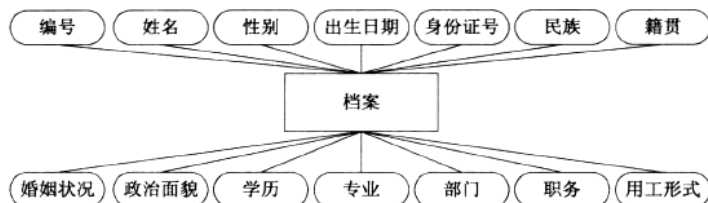


图 17.9 档案信息表 E-R 图

本系统提供了人事考勤记录和人事奖惩记录功能，这里只给出人事考勤信息表的 E-R 图，如图 17.10 所示。

根据企业人事管理中的现实需求，本系统提供了多账套管理功能，通过这一功能，可以很方便地对各种类型的员工实施不同的待遇标准。账套信息表的 E-R 图如图 17.11 所示。

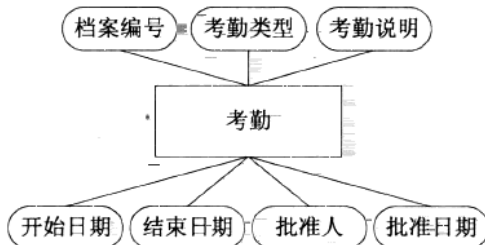


图 17.10 考勤信息表 E-R 图

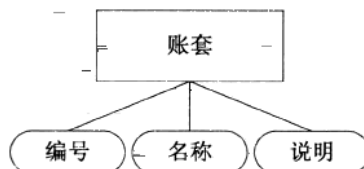


图 17.11 账套信息表 E-R 图

每个账套都要包含多个账套项目，这些账套中的项目可以有零个或多个是不同的，区别是每个账套项目的金额是不同的。账套项目信息表的 E-R 图如图 17.12 所示。

建立多账套是为了实现对员工按照不同的待遇标准进行管理，所以要将员工设置到不同的账套中，即表示对该员工实施相应的待遇标准。账套设置信息表的 E-R 图如图 17.13 所示。

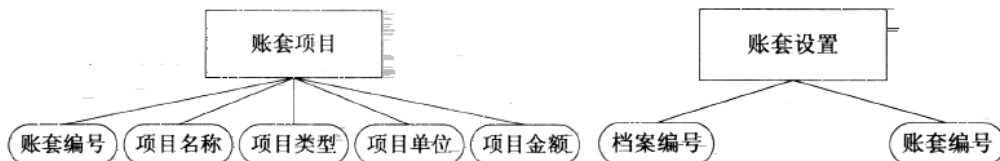


图 17.12 账套项目信息表 E-R 图

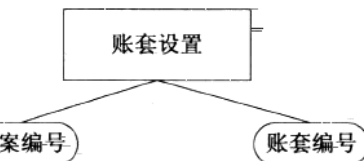


图 17.13 账套设置信息表 E-R 图

### 17.4.3 数据库逻辑结构设计

数据库概念设计中已经分析了档案、考勤和账套等主要的数据库实体对象，这些实体对象是数据库表结构的基本模型，最终的数据模型都要实施到数据库中，形成整体的数据库结构。可以使用 PowerDesigner 工具完成这个数据库的建模，其模型结构如图 17.14 所示。

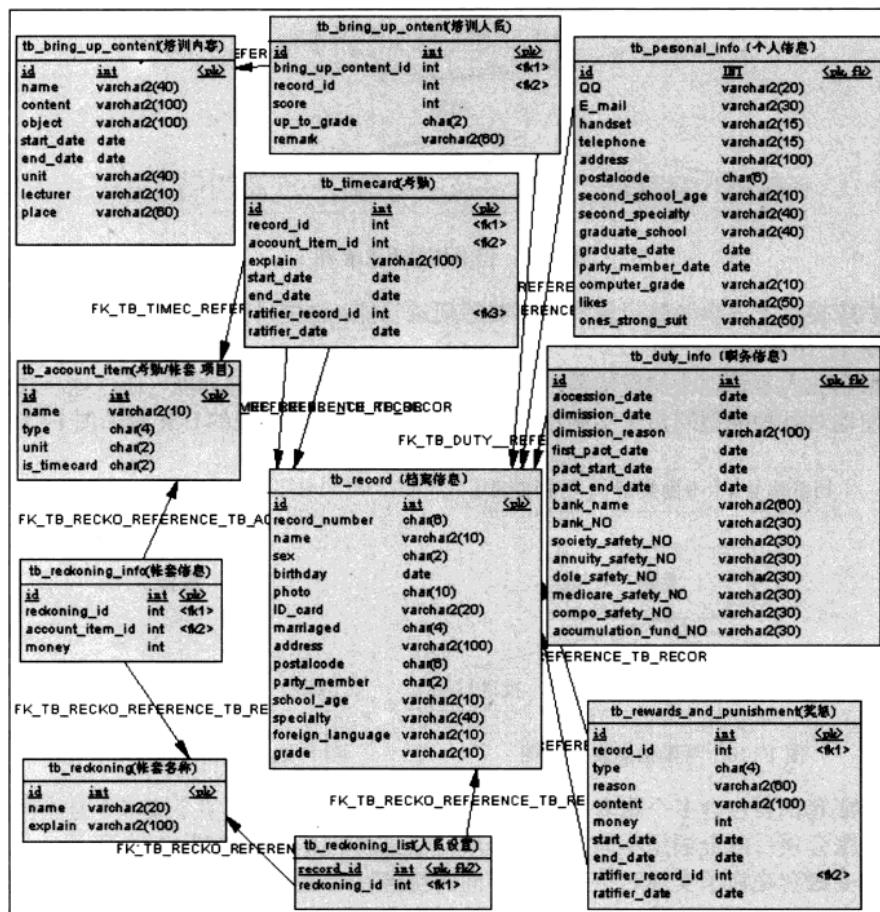
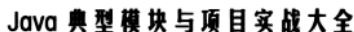


图 17.14 企业人事管理系统数据库模型

主窗体是软件系统的一个重要组成部分，是提供人机交互的一个必不可少的操作平台。通过主窗体，用户可以打开与系统相关的各个子操作模块，完成对软件的操作和使用；另外通过主窗体，用户还可以快速掌握本系统的基本功能。

通过本系统的导航栏,可以打开本系统的所有子模块。导航栏的效果如图 17.15 所示。



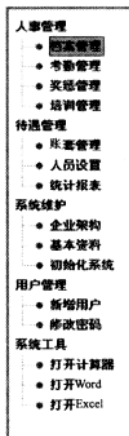


图 17.15 导航栏效果

本系统的导航栏是通过树组件实现的, 在这里不显示树的根节点, 并且打开软件时树结构是展开的, 还设置在叶子节点折叠和展开时均不采用图标。下面的代码将通过树节点对象创建一个树结构, 最后创建一个树模型对象。

```
//创建树的根节点
DefaultMutableTreeNode root = new DefaultMutableTreeNode("root");
//创建树的一级子节点
DefaultMutableTreeNode personnelNode = new DefaultMutableTreeNode("人事管理");
//创建树的叶子节点并添加到一级子节点
personnelNode.add(new DefaultMutableTreeNode("档案管理"));
personnelNode.add(new DefaultMutableTreeNode("考勤管理"));
personnelNode.add(new DefaultMutableTreeNode("奖惩管理"));
personnelNode.add(new DefaultMutableTreeNode("培训管理"));
root.add(personnelNode); //向根节点添加一级子节点
DefaultMutableTreeNode treatmentNode = new DefaultMutableTreeNode("待遇管理");
treatmentNode.add(new DefaultMutableTreeNode("工资管理"));
treatmentNode.add(new DefaultMutableTreeNode("人员设置"));
treatmentNode.add(new DefaultMutableTreeNode("统计报表"));
root.add(treatmentNode);
DefaultMutableTreeNode systemNode = new DefaultMutableTreeNode("系统维护");
systemNode.add(new DefaultMutableTreeNode("企业架构"));
systemNode.add(new DefaultMutableTreeNode("基本资料"));
systemNode.add(new DefaultMutableTreeNode("初始化系统"));
root.add(systemNode);
DefaultMutableTreeNode userNode = new DefaultMutableTreeNode("用户管理");
//当 record 为 null 时, 说明是通过默认用户登录的, 此时只能新增用户, 不能修改密码
if (record == null) {
    userNode.add(new DefaultMutableTreeNode("新增用户"));
} else {
    String purview = record.getTbManager().getPurview(); //否则为通过管理员登录
    //只有当管理员的权限为“超级管理员”时, 才有权新增用户
    if (purview.equals("超级管理员")) {
        userNode.add(new DefaultMutableTreeNode("新增用户"));
    }
    //只有通过管理员登录时才有修改密码
    userNode.add(new DefaultMutableTreeNode("修改密码"));
}
root.add(userNode);
DefaultMutableTreeNode toolNode = new DefaultMutableTreeNode("系统工具");
toolNode.add(new DefaultMutableTreeNode("打开计算器"));
toolNode.add(new DefaultMutableTreeNode("打开 Word"));
toolNode.add(new DefaultMutableTreeNode("打开 Excel"));
```





```

root.add(toolNode);
//通过树节点对象创建树模型对象
DefaultTreeModel treeModel = new DefaultTreeModel(root);

```

下面的代码将利用在上述例程中创建的树模型对象创建一个树对象,并设置树对象的相关绘制属性。

```

tree = new JTree(treeModel); //通过树模型对象创建树对象
tree.setBackground(Color.WHITE); //设置树的背景色
❶ tree.setRootVisible(false); //设置不显示树的根节点
tree.setRowHeight(28); //设置各节点的高度为 28 像素
Font font = new Font("宋体", Font.BOLD, 16);
tree.setFont(font); //设置节点的字体样式
//创建一个树的绘制对象
DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
renderer.setClosedIcon(null); //设置节点折叠时不采用图标
renderer.setOpenIcon(null); //设置节点展开时不采用图标
tree.setCellRenderer(renderer); //将树的绘制对象设置到树中
int count = root.getChildCount(); //获得一级节点的数量
for (int i = 0; i < count; i++) { //遍历树的一级节点
    //获得指定索引的一级节点对象
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) root.getChildAt(i);
    TreePath path = new TreePath(node.getPath()); //获得节点对象的路径
    ❷ tree.expandPath(path); //展开该节点
}
//捕获树的选取事件
❸ tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        .....//由于篇幅有限,此处省略了处理捕获事件的具体代码,详见光盘源代码
    }
});
leftPanel.add(tree); //将树添加到面板组件中
}

```

### 🔊 关键代码解析

❶ `tree.setRootVisible(boolean b)`: 该方法用于设置是否显示树的根节点,默认为显示根节点,即默认为 `true`; 如果设置为 `false`, 则不显示树的根节点。

❷ `tree.expandPath(TreePath path)`: 该方法用于展开指定路径的节点。

❸ `tree.addTreeSelectionListener(TreeSelectionListener listener)`: 该方法用于为树添加捕获树节点被选中 and 取消的事件。

## 17.5.2 工具栏的设计

为了方便用户使用系统,在工具栏为常用的系统子模块提供了快捷按钮,通过这些按钮,用户可以快速地进入系统中常用的子模块。工具栏的效果如图 17.16 所示。



图 17.16 工具栏效果

下面的代码将创建一个用来添加快捷按钮的面板,并且为面板设置了边框,面板的布







局管理器为水平箱式布局。

```
final JPanel buttonPanel = new JPanel(); //创建工具栏面板
//创建一个水平箱式布局管理器对象
final GridLayout gridLayout = new GridLayout(1, 0);
gridLayout.setVgap(6); //箱的垂直间隔为 6 像素
gridLayout.setHgap(6); //箱的水平间隔为 6 像素
buttonPanel.setLayout(gridLayout); //设置工具栏面板采用的布局管理器为箱式布局
buttonPanel.setBackground(Color.WHITE); //设置工具栏面板的背景色
buttonPanel.setBorder(new TitledBorder(null, "", TitledBorder.DEFAULT_JUSTIFICATION,
    TitledBorder.DEFAULT_POSITION, null, null)); //设置工具栏面板采用的边框样式
topPanel.add(buttonPanel, BorderLayout.CENTER); //将工具栏面板添加到上级面板中
```

在工具栏中提供了用来快速打开“档案管理”、“考勤管理”、“奖惩管理”、“统计报表”、“基本资料”和“修改密码”子模块的按钮,以及“打开计算器”和“打开 Excel”两个打开常用系统工具的按钮,还有一个用来快速退出系统的“退出”按钮。这些快捷按钮的实现代码基本相同,所以这里只给出“档案管理”快捷按钮的实现代码,具体代码如下:

```
//创建进入“档案管理”的快捷按钮
final JButton recordShortcutKeyButton = new JButton();
//为按钮添加事件监听器,用来捕获按钮被单击的事件
recordShortcutKeyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        rightPanel.removeAll(); //移除内容面板中的所有内容
        rightPanel.add(new RecordSelectedPanel(rightPanel),
            BorderLayout.CENTER); //将档案管理面板添加到内容面板中
        SwingUtilities.updateComponentTreeUI(rightPanel); //刷新内容面板中的内容
    }
});
recordShortcutKeyButton.setText("档案管理");
buttonPanel.add(recordShortcutKeyButton);
```

在实现“修改密码”按钮时,需要判断当前的登录用户,如果用户是通过系统的默认用户登录的,则不允许修改密码,需要把“修改密码”按钮设置为不可用。具体代码如下:

```
final JButton updatePasswordShortcutKeyButton = new JButton();
if (record == null) //当 record 为 null 时,说明是通过默认用户登录的,此时不能修改密码
//在这种情况下“设置”按钮为不可用
updatePasswordShortcutKeyButton.setEnabled(false);
updatePasswordShortcutKeyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        rightPanel.removeAll();
        SwingUtilities.updateComponentTreeUI(rightPanel);
        //创建用来修改密码的对话框
        UpdatePasswordDialog dialog = new UpdatePasswordDialog();
        dialog.setRecord(record); //将当前登录管理员的档案对象传入对话框
        dialog.setVisible(true); //设置对话框为可见的,即显示对话框
    }
});
updatePasswordShortcutKeyButton.setText("修改密码");
buttonPanel.add(updatePasswordShortcutKeyButton);
```

通过 java.awt.Desktop 类的 open(File file) 方法,可以运行系统中的其他软件,例如,运行系统计算器。为了方便用户使用系统计算器和 Excel,本系统提供了“打开计算器”和“打开 Excel”两个按钮。这两个按钮的实现代码基本相同,下面将以打开系统计算器为例,讲解如何在 Java 程序中打开其他软件。具体代码如下:

```
final JButton counterShortcutKeyButton = new JButton();
counterShortcutKeyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Desktop desktop = Desktop.getDesktop(); //获得当前系统对象
        //创建一个系统计算器对象
        File file = new File("C:/WINDOWS/system32/calc.exe");
```







```
        try {
            desktop.open(file); //打开系统计算器
        } catch (Exception e) { //当打开失败时，弹出提示信息
            JOptionPane.showMessageDialog(null, "很抱歉，未能打开系统自带的计算器！",
                "友情提示", JOptionPane.INFORMATION_MESSAGE);
            return;
        }
    });
    counterShortcutKeyButton.setText("打开计算器");
    buttonPanel.add(counterShortcutKeyButton);
```

最后，创建一个用来快速退出系统的“退出”按钮，具体代码如下：

```
final JButton exitShortcutKeyButton = new JButton();
exitShortcutKeyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0); //退出系统
    }
});
exitShortcutKeyButton.setText("退出");
buttonPanel.add(exitShortcutKeyButton);
```

## 17.6 公共模块设计

公共模块的设计是软件开发的一个重要组成部分，它既起到了代码重用的作用，又起到了规范代码结构的作用，尤其在团队开发的情况下，是解决重复编码的最好方法，这样对软件的后期维护也起到了积极的作用。

### 17.6.1 编写 Hibernate 配置文件

在 Hibernate 配置文件中包含两方面的内容，一方面是连接数据库的基本信息，例如，连接数据库的驱动程序、URL、用户名、密码等；另一方面是 Hibernate 的配置信息，例如，配置数据库使用的方言、持久化类映射文件等，还可以配置是否在控制台输出 SQL 语句，以及是否对输出的 SQL 语句进行格式化和添加提示信息等。本系统使用的 Hibernate 配置文件的关键代码如下：

```
<property name="connection.driver_class">com.microsoft.jdbc.sqlserver.SQLServerDriver</property> <!-- 配置数据库的驱动类 -->
<property name="connection.url">jdbc:sqlserver://localhost:1433;databasename=db_PersonnelManage</property> <!-- 配置数据库的连接路径 -->
<property name="connection.username">sa</property> <!-- 配置数据库的连接用户名 -->
<property name="connection.password"></property> <!-- 配置数据库的连接密码 -->
<property name="dialect">org.hibernate.dialect.SQLServerDialect</property> <!-- 配置数据库使用的方言 -->
<property name="show_sql">true</property> <!-- 配置在控制台显示 SQL 语句 -->
<property name="format_sql">true</property> <!-- 配置对输出的 SQL 语句进行格式化 -->
<!-- 配置在输出的 SQL 语句前面添加提示信息 -->
```







```
<property name="use_sql_comments">true</property>
<!-- 配置持久化类映射文件 -->
❶ <mapping resource="com/mwq/hibernate/mapping/TbDept.hbm.xml" />
```

### 🔊 关键代码解析

❶ connection.password: 该属性用来配置连接数据库的密码, 这里密码为空, 在这种情况下也可以省略该行配置代码, 这里加上是为了讲解之用。

❷ show\_sql: 该属性用来配置是否在控制台输出 SQL 语句, 默认为 false, 即不输出; 建议在调试程序时将该属性及 format\_sql 和 use\_sql\_comments 属性同时设置为 true, 这样可以快速找出错误原因, 但是在发布程序之前, 一定要将这 3 个属性再设置为 false (也可以删除这 3 行配置代码, 因为它们的默认值均为 false, 笔者推荐删除), 这样做的好处是节省了格式化、注释和输出 SQL 语句的时间, 从而提高了软件的性能。

❸ resource: 该属性用来配置持久化类映射文件, 每一个持久化类都要做这样的映射, 由于篇幅有限, 这里只给出了一行, 详见光盘源代码。

## 17.6.2 编写 Hibernate 持久化类和映射文件

持久化类是数据实体的对象表现形式, 通常情况下持久化类与数据表是相互对应的, 它们通过持久化类映射文件建立映射关系。持久化类不需要实现任何类和接口, 只需要提供一些属性及其对应的 set/get 方法。需要注意的是, 每一个持久化类都需要提供一个没有入口参数的构造方法。

下面是持久化类 TbRecord 的部分代码, 为了节省篇幅, 这里只给出了两个具有代表性的属性, 其中属性 id 为主键。

```
public class TbRecord {
    public TbRecord () {
    }
    private int id;
    private String name;
    public void setId(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

下面是与持久化类 TbRecord 对应的映射文件 TbRecord.hbm.xml 的相应代码, 持久化类映射文件负责建立持久化类与对应数据表之间的映射关系。

```
❶ <class name="com.mwq.hibernate.mapping.TbRecord" table="tb_record"
    schema="dbo" catalog="db_PersonnelManage">
❷ <id name="id" type="java.lang.Integer">
    <column name="id" />
❸ <generator class="increment" />
```





```
</id>
① <property name="name" type="java.lang.String">
② <column name="name" length="10" not-null="true" />
</property>
</class>
```

### 🔊 关键代码解析

① <class>: 该元素用来配置持久化类与数据表之间的映射关系, 其中, name 属性为持久化类名称, table 属性为数据表名称, schema 属性为数据表所属用户的权限, catalog 属性为数据表所在的数据库名称。

② <id>: 该元素用来配置主键的映射关系, 其中 name 属性为持久化类中属性的名称, type 属性为持久化类中属性的类型。

③ <generator>: 该元素用来配置主键的生成方式, 当将 class 属性设置为 increment 时, 表示采用 Hibernate 自增。

④ <property>: 该元素用来配置属性的映射关系, 其中, name 属性为持久化类中属性的名称, type 属性为持久化类中属性的类型。

⑤ <column>: 该元素用来配置与持久化类中的属性对应的列, 其中, name 属性为列的名称, length 属性为值的最大长度, not-null 属性为是否允许为空, 当设为 true 时表示不允许为空, 默认为 false。

## 17.6.3 编写通过 Hibernate 操作持久化对象的常用方法

对数据库的操作, 离不开增、删、改、查, 所以在这里也离不开实现这些功能的方法。不过在这里还针对 Hibernate 的特点, 实现了两个具有特殊功能的方法, 一个是用来过滤关联对象集合的方法, 另一个是用来批量删除记录的方法。下面只介绍这两个方法和一个删除单个对象的方法, 其他参见光盘源代码。

下面的方法是用来过滤一对多关联中 Set 集合中的对象的方法, 这是 Hibernate 提供的一个非常实用的集合过滤功能, 通过该功能可以从关联集合中检索出符合指定条件的对象, 检索条件可以是所有合法的 HQL 语句, 具体代码如下:

```
public List filterSet(Set set, String hql) {
    Session session = HibernateSessionFactory.getSession(); //获得 Session 对象
    //通过 Session 对象的 createFilter() 方法按照 HQL 条件过滤 set 集合
    Query query = session.createFilter(set, hql);
    List list = query.list(); //执行过滤, 返回值为 List 型结果集
    return list; //返回过滤结果
}
```

下面的方法用来删除指定持久化对象, 具体代码如下。

```
public boolean deleteObject(Object obj) {
    boolean isDelete = true; //默认删除成功
    //获得 Session 对象
    Session session = HibernateSessionFactory.getSession();
    Transaction tr = session.beginTransaction(); //开启事务
    try {
        session.delete(obj); //删除指定持久化对象
        tr.commit(); //提交事务
    } catch (HibernateException e) {
        isDelete = false; //删除失败
    }
}
```







```

        tr.rollback(); //回退事务
        e.printStackTrace();
    }
    return isDelete;
}

```

下面的方法用来批量删除对象,通过这种方法删除对象,每次只需要执行一条 SQL 语句,具体代码如下:

```

public boolean deleteOfBatch(String hql) {
    boolean isDelete = true; //默认删除成功
    Session session = HibernateSessionFactory.getSession(); //获得 Session 对象
    Transaction tr = session.beginTransaction(); //开启事务
    try {
        Query query = session.createQuery(hql); //预处理 HQL 语句,获得 Query 对象
        query.executeUpdate(); //执行批量删除
        tr.commit(); //提交事务
    } catch (HibernateException e) {
        isDelete = false; //删除失败
        tr.rollback(); //回退事务
        e.printStackTrace();
    }
    return isDelete;
}

```

#### 17.6.4 创建用于特殊效果的部门树对话框

在系统中有多处需要填写部门,如果通过 JComboBox 组件提供部门列表,则不能够体现出企业的组织架构,用户在使用过程中也不是很直观和方便,因此开发了一个用于特殊效果的部门树对话框,例如,在新建档案时需要填写部门,利用该对话框实现的效果如图 17.17 所示。

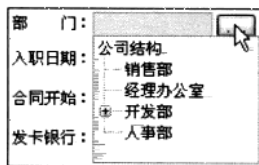


图 17.17 用于特殊效果的部门树对话框

用户在填写部门时,只需要单击文本框后的按钮,就会弹出一个用来选取部门的部门树对话框,并且这个对话框显示在文本框和按钮的正下方,通过这种方法实现对部门的选取,对于用户将更加直观和方便。

从图 17.17 可以看出,这里用来选取部门的部门树对话框不需要提供标题栏,并且建议令这个对话框阻止当前线程,这样做的好处是可以强制用户选取部门,并且可以及时地销毁对话框,释放其占用的资源。实现这两点的具体代码如下:

```

setModal(true); //设置对话框阻止当前线程
setUndecorated(true); //设置对话框不提供标题栏

```

下面开始创建部门树。首先创建树节点对象,包括根节点及其子节点,并将子节点添加到上级节点中,然后利用根节点对象创建树模型对象,最后利用树模型对象创建树对象。当树节点超过一定数量时,树结构的高度可能大于对话框的高度,所以要将部门树放在滚动面板当中。具体代码如下:

```

final JScrollPane scrollPane = new JScrollPane(); //创建滚动面板
getContentPane().add(scrollPane, BorderLayout.CENTER);
TbDept company = (TbDept) dao.queryDeptById(1);
//创建部门树的根节点
DefaultMutableTreeNode root = new DefaultMutableTreeNode(company.getName());

```



```

Set depts = company.getTbDepts();
for (Iterator deptIt = depts.iterator(); deptIt.hasNext();) {
    TbDept dept = (TbDept) deptIt.next();
    DefaultMutableTreeNode deptNode = new DefaultMutableTreeNode
        (dept.getName()); //创建部门树的二级子节点
    root.add(deptNode);
    Set sonDepts = dept.getTbDepts();
    for (Iterator sonDeptIt = sonDepts.iterator(); sonDeptIt.hasNext();) {
        TbDept sonDept = (TbDept) sonDeptIt.next();
        //创建部门树的叶子节点
        deptNode.add(new DefaultMutableTreeNode(sonDept.getName()));
    }
}
//利用根节点对象创建树模型对象
DefaultTreeModel treeModel = new DefaultTreeModel(root);
tree = new JTree(treeModel); //利用树模型对象创建树对象
scrollPane.setViewportView(tree); //将部门树放到滚动面板中

```

在通过构造方法创建部门树对话框时,需要传入要填写部门的文本框对象,这样在捕获树节点被选中的事件后会自动填写部门名称。用来捕获树节点被选中事件的具体代码如下:

```

//捕获树节点被选中的事件
tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        TreePath treePath = e.getPath(); //获得被选中树节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) treePath
            .getLastPathComponent(); //获得被选中树节点的对象
        if (node.getChildCount() == 0) { //被选中的节点为叶子节点
            textField.setText(node.toString()); //将选中节点的名称显示到传入的文本框中
        } else { //被选中的节点不是叶子节点
            JOptionPane.showMessageDialog(null, "请选择所在的具体部门",
                "错误提示", JOptionPane.ERROR_MESSAGE);
            return;
        }
        dispose(); //销毁部门树对话框
    }
});

```

### 17.6.5 创建通过部门树选取员工的面板和对话框

在系统中有多处需要通过部门树选取员工,其中一处是在主窗体中,其他的均在对话框中,所以,这里需要单独实现一个通过部门树选取员工的面板,然后将面板添加到主窗体或对话框中,从而实现代码的最大重用。最终实现的对话框效果如图 17.18 所示,当选中左侧部门树中的相应部门时,在右侧表格中将列出该部门及其子部门的所有员工。

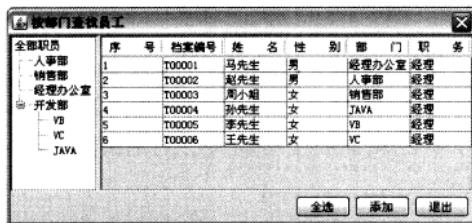


图 17.18 “按部门查找员工”对话框

下面实现面板类 DeptAndPersonnelPanel。







(1) 创建表格, 在创建表格时, 可以通过向量初始化表格, 也可以通过数组初始化表格。具体代码如下:

```
tableColumnV = new Vector<String>(); //创建表格列名向量
String tableColumns[] = new String[] { "序号", "档案编号", "姓名", "性别", "部门", "职务" };
for (int i = 0; i < tableColumns.length; i++) { //添加表格列名
    tableColumnV.add(tableColumns[i]);
}
tableValueV = new Vector<Vector<String>>(); //创建表格值向量
showAllRecord(); //默认显示所有档案
//创建表格模型对象
tableModel = new DefaultTableModel(tableValueV, tableColumnV);
table = new JTable(tableModel); //创建表格对象
personnalScrollPane.setViewportViewView(table); //将表格添加到滚动面板中
```

(2) 为部门树添加节点选取事件处理代码, 当选取的为根节点时, 将显示所有档案, 当选取的为子节点时, 将显示该部门的档案, 否则显示选中部门包含子部门的所有档案。具体代码如下:

```
tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        TreePath path = e.getPath(); //获得被选中树节点的路径
        tableValueV.removeAllElements(); //移除表格中的所有行
        if (path.getPathCount() == 1) { //选中树的根节点
            showAllRecord(); //显示所有档案
        } else { //选中树的子节点
            //获得选中部门的名称
            String deptName = path.getLastPathComponent().toString();
            //检索指定部门对象
            TbDept selectDept = (TbDept) dao.queryDeptByName(deptName);
            Iterator sonDeptIt = selectDept.getTbDepts().iterator();
            if (sonDeptIt.hasNext()) { //选中树的二级节点
                while (sonDeptIt.hasNext()) {
                    //显示选中部门所有子部门的档案
                    showRecordInDept((TbDept) sonDeptIt.next());
                }
            } else { //选中树的叶子节点
                showRecordInDept(selectDept); //显示选中部门的档案
            }
        }
        tableModel.setDataVector(tableValueV, tableColumnV);
    }
});
```

下面实现对话框类 DeptAndPersonnelDialog, 在对话框中提供 3 个按钮, 用户可以通过“全选”按钮选择表格中的所有档案, 也可以用鼠标点选指定档案, 然后单击“添加”按钮, 将选中的档案记录添加到指定向量中, 添加结束后单击“退出”按钮。需要注意的是, 在单击“退出”按钮时并没有销毁对话框, 只是将其变为不可见, 在调用对话框的位置获得选中档案信息之后才销毁对话框。负责捕获“添加”按钮事件的具体代码如下:

```
final JButton addButton = new JButton();
addButton.addActionListener(new ActionListener() { //捕获按钮被按下的事件
    public void actionPerformed(ActionEvent e) {
        int[] rows = table.getSelectedRows(); //获得选中行的索引
        int columnCount = table.getColumnCount(); //获得表格的列数
        for (int row = 0; row < rows.length; row++) {
            //创建一个向量对象, 代表表格的一行
            Vector<String> recordV = new Vector<String>();
            for (int column = 0; column < columnCount; column++) {
                recordV.add(table.getValueAt(rows[row], column).toString());
            }
            //将表格中的值添加到向量中
        }
    }
});
```





```
selectedRecordV.add(recordV); //将代表选中行的向量添加到另一个向量中  
}  
});  
addButton.setText("添加"); //设置按钮的名称
```

## 17.7 人事管理模块设计

人事管理模块是企业人事管理系统的灵魂,是其他模块的基础,所以能否合理设计人事管理模块,对系统的整体设计和系统功能的开发将起到十分重要的作用。

### 17.7.1 人事管理模块功能概述

人事管理模块包含档案管理、考勤管理、奖惩管理和培训管理 4 个子模块。

档案管理模块用来建立和修改员工档案,当进入档案管理模块时,将看到如图 17.19 所示的界面。

		新建员工档案		修改员工档案			
全部职员		序 号	档案编号	姓 名	性 别	部 门	职 务
销售部		1	T00001	马先生	男	经理办公室	经理
开发部		2	T00002	赵先生	男	人事部	经理
VB		3	T00003	周小姐	女	销售部	经理
JAVA		4	T00004	孙先生	男	JAVA	经理
VC		5	T00005	李先生	男	VB	经理

图 17.19 员工档案列表界面

单击“新建员工档案”按钮或在表格中选中要修改的员工档案后单击“修改员工档案”按钮,将打开如图 17.20 所示的界面,在该界面可以建立或修改员工档案,并且可以设置员工照片,填写完成后单击“保存”按钮保存员工档案。

档案编号: T00001		保存 提交	
基本信息			
姓名: 马先生	性别: <input checked="" type="radio"/> 男 <input type="radio"/> 女	出生日期: 1980-01-05	
民族: 汉族	籍贯: 吉林省	身份证: 220101	
学历: 本科	专业: 计算机	政治面貌: 通	
外语语种: 英语	外语水平: 四级	婚姻状况: <input checked="" type="radio"/> 未婚 <input type="radio"/> 已婚	
邮政编码: 130000		户籍地址: 吉林省长春市	
任职信息			
部门: 经理办公室	职务: 经理	入职日期: 2006-08-08	入职类型: 全职工
合同开始: 2006-08-08	合同结束: 2011-08-08	转正日期: 2007-08-08	转正类型: 转正工
发卡银行: 工商银行	社会保险: 医疗保险	失业保险: 失业保险	养老保险: 养老保险
信用卡号: 1234567890123456	医疗保险: 医疗保险	工伤保险: 工伤保险	公积金号: 1234567890123456
个人信息			
移动电话: 13800000000	固定电话: 0431-12345678	QQ: 12345678	E-mail: 12345678@163.com
第二学历: 第二专业: 英语	毕业日期: 2006-08-08	毕业学校: 吉林大学	
电子邮箱: 1234567890@163.com	党 籍: 中共党员	特 别: 无	
入职日期: 2006-08-08	家庭住址: 吉林省长春市		

图 17.20 填写档案信息界面

考勤管理和奖惩管理模块用来填写相关记录,这些记录信息将体现在统计报表模块,例如,在这里给某员工填写一次迟到考勤,在做统计报表时将根据其采用的账套在其待遇







中扣除相应的金额。这两个模块的实现思路基本相同,在这里只给出考勤管理界面,如图 17.21 所示。

图 17.21 考勤管理界面

培训管理模块用来记录对员工的培训信息,如图 17.22 所示,选中培训记录后单击“查看”按钮可以查看具体的培训人员。

序号	培训名称	培训对象	参训人数	培训时间	培训地点	培训内容	培训单位	培训讲师
1	入职培训	新员工	5	2007-01-13	单位会议室	入职基本培训	本单位	马先生
2	领域大厦项目培训	相关人员	5	2007-01-14	单位会议室	领域大厦项目...	本单位	马先生

图 17.22 培训列表界面

## 17.7.2 人事管理模块技术分析

在开发该模块时,需要处理大量用户输入的信息。处理用户输入信息的第一步是检查用户输入信息的合法性。如果是利用常规方法去验证每个组件接收到的数据,将耗费大量的时间和代码。对于这种情况,可以利用 Java 的反射机制先进行简单的验证,例如,不允许为空的验证,然后再针对特殊的数据进行具体的验证,例如,日期型数据。

在建立员工档案时需要支持上传员工照片的功能。如果想支持这一功能,必须了解两项关键技术,一是如何弹出用来选取照片的对话框,二是如何将照片文件上传到指定的位置。用来选取照片的对话框可以通过 javax.swing.JFileChooser 类实现,还可以通过实现 javax.swing.filechooser.FileFilter 接口,对指定路径中的文件进行过滤,令照片选取对话框中只显示照片文件。实现上传照片功能需要通过 java.io.File、java.io.FileInputStream 和 java.io.FileOutputStream 类联合实现。

在考勤管理和奖惩管理模块,既可以直接在员工下拉列表框中选取考勤或奖惩的员工,也可以先选取员工所在的部门,对员工下拉列表框中的可选项进行筛选,然后再选取具体的员工。要实现这一功能,需要实现组件之间的联动,即当选取部门时,将间接控制员工下拉列表框的变化;同样在选取员工下拉列表框时,也要间接控制部门组件的变化,即在部门组件中要显示员工所在的部门。可以通过捕获各个组件的事件完成这一功能,例如,通过 java.awt.event.ItemListener 监听器捕获下拉列表框中被选中的事件,通过 javax.swing.event.TreeSelectionListener 监听器捕获树节点被选中的事件。





### 17.7.3 人事管理模块实现过程

■ 人事管理使用的主要数据表包括 tb\_record、tb\_duty\_info、tb\_personal\_info、tb\_timecard、tb\_rewards\_and\_punishment、tb\_bring\_up\_content、tb\_bring\_up\_content。

在开发人事管理模块时,主要是突破技术分析中的几个技术点,掌握这几个技术点后,就可以顺利地实现人事管理模块了。

#### 1. 实现上传员工照片功能

(1) 在开发上传员工照片功能时,首先是确定显示照片的载体。在 Swing 中可以通过 JLabel 组件显示照片,在该组件中也可以显示文字。在本系统中如果已上传照片则显示照片,否则显示提示文字。具体代码如下:

```
photoLabel = new JLabel(); //创建用来显示照片的对象
//设置照片或文字居中显示
photoLabel.setHorizontalAlignment(SwingConstants.CENTER);
photoLabel.setBorder(new TitledBorder(null, "",
TitledBorder.DEFAULT_JUSTIFICATION, TitledBorder.DEFAULT_POSITION, null, null));
//设置边框
photoLabel.setPreferredSize(new Dimension(120, 140)); //设置显示照片的大小
//新建档案或未上传照片
if (UPDATE_RECORD == null || UPDATE_RECORD.getPhoto() == null) {
    photoLabel.setText("双击添加照片"); //显示文字提示
} else { //修改档案并且上传照片
    //获得指定路径的绝对路径
    URL url = this.getClass().getResource("/personnel_photo/");
    //组织员工照片的存放路径
    String photo = url.toString().substring(5) + UPDATE_RECORD.getPhoto();
    photoLabel.setIcon(new ImageIcon(photo)); //创建照片对象并显示
}
```

(2) 确定如何弹出供用户选取照片的对话框。可以通过按钮捕获用户上传照片的请求,也可以通过 JLabel 组件自己捕获该请求,即为 JLabel 组件添加鼠标监听器,当用户双击该组件时,弹出供用户选取照片的对话框,本系统采用的是后者。Swing 提供了一个用来选取文件的对话框类 JFileChooser,当执行 JFileChooser 类的 showOpenDialog()方法时将弹出文件选取对话框,如图 17.23 所示。该方法返回 int 型值,用来区别用户执行的操作。当返回值为静态常量 APPROVE\_OPTION 时,表示用户选取了照片,在这种情况下将选中的照片显示到 JLabel 组件中。具体代码如下:

```
photoLabel.addMouseListener(new MouseAdapter() { //添加鼠标监听器
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) { //判断是否为双击
            JFileChooser fileChooser = new JFileChooser(); //创建文件选取对话框
            fileChooser.setFileFilter(new FileFilter() { //为对话框添加文件过滤器
                public String getDescription() { //设置提示信息
                    return "图像文件 (.jpg;.gif)";
                }
                public boolean accept(File file) { //设置接收文件类型
                    if (file.isDirectory())
                        return true; //为文件夹则返回 true
                    String fileName = file.getName().toLowerCase();
                    if (fileName.endsWith(".jpg") || fileName.endsWith(".gif"))
                        return true; //为 JPG 或 JIF 格式文件则返回 true
                    //否则返回 false,即不显示在文件选取对话框中
                    return false;
                }
            });
            int result = fileChooser.showOpenDialog(photoLabel);
            if (result == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                String photoPath = photoLabel.getIcon().getIconFile().getPath();
                photoLabel.setIcon(new ImageIcon(selectedFile.getPath()));
            }
        }
    }
});
```







```

    });
    //弹出文件选取对话框并接收用户的处理信息
    int i = fileChooser.showOpenDialog(getParent());
    if (i == fileChooser.APPROVE_OPTION) { //用户选取了照片
        File file = fileChooser.getSelectedFile(); //获得用户选取的文件对象
        if (file != null) {
            ImageIcon icon = new ImageIcon(file
                .getAbsolutePath()); //创建照片对象
            photoLabel.setText(null); //取消提示文字
            photoLabel.setIcon(icon); //显示照片
        }
    }
}
});

```



图 17.23- 选取照片

(3) 在保存档案信息时将照片上传到指定路径下, 上传到指定路径下的照片名称将修改为档案编号, 但是因为可以上传两种格式的图片, 为了记录图片格式, 还是要将照片名称保存到数据库中。具体代码如下:

```

if (photoLabel.getIcon() != null) { //查看是否上传照片
    //通过选中图片的路径创建文件对象
    File selectPhoto = new File(photoLabel.getIcon().toString());
    //获得指定路径的绝对路径
    URL url = this.getClass().getResource("/personnel_photo/");
    StringBuffer uriBuffer = new StringBuffer(url.toString()); //组织文件路径
    String selectPhotoName = selectPhoto.getName();
    int i = selectPhotoName.lastIndexOf(".");
    uriBuffer.append(recordNoTextField.getText());
    uriBuffer.append(selectPhotoName.substring(i));
    try {
        //创建上传文件对象
        File photo = new File(new URL(uriBuffer.toString()).toURI());
        record.setPhoto(photo.getName()); //将图片名称保存到数据库
        if (!photo.exists()) { //如果文件不存在则创建文件
            photo.createNewFile();
        }
        InputStream inStream = new FileInputStream(selectPhoto); //创建输入流对象
        OutputStream outStream = new FileOutputStream(photo); //创建输出流对象
        int readBytes = 0; //读取字节数
        byte[] buffer = new byte[10240]; //定义缓存数组
        //从输入流读取数据到缓存数组中
        while ((readBytes = inStream.read(buffer, 0, 10240)) != -1) {
            outStream.write(buffer, 0, readBytes); //将缓存数组中的数据输出到输出流
        }
        outStream.close(); //关闭输出流对象
        inStream.close(); //关闭输入流对象
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```



523





## 2. 实现组件联动功能

在开发考勤功能时,实现了部门和员工组件之间的联动功能,如果用户直接单击“考勤员工”下拉列表框,在下拉列表框中将显示所有员工,如图 17.24 所示,这是因为在初始化“考勤员工”下拉列表框时,添加的是所有员工。具体代码如下:

```
personalComboBox = new JComboBox(); //创建下拉列表框对象
personalComboBox.addItem("请选择"); //添加提示项
Iterator recordIt = dao.queryRecord().iterator(); //检索所有员工
while (recordIt.hasNext()) { //通过循环添加到下拉列表框中
    TbRecord record = (TbRecord) recordIt.next();
    personalComboBox.addItem(record.getRecordNumber() + " " +
record.getName());
}
```

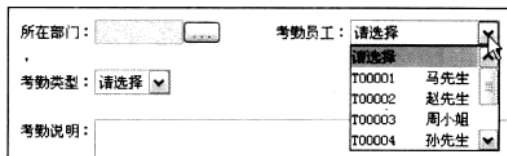


图 17.24 单击“考勤员工”下拉列表框

当用户选中考勤员工后,在“所在部门”文本框将填入选中员工所在的部门,实现这一功能是通过捕获下拉列表框选项状态发生改变的事件,具体代码如下:

```
//捕获下拉列表框选项状态发生改变的事件
personalComboBox.addItemListener(new ItemListener() {
    ❶ public void itemStateChanged(ItemEvent e) {
        //查看是否是由选中当前项触发的
        ❷ if (e.getStateChange() == ItemEvent.SELECTED) {
            ❸ String selectedItem = (String) e.getItem(); //获得选中项的内容
            //当选项为“请选择”时,设置部门文本框为空
            if (selectedItem.equals("请选择")) {
                inDeptTextField.setText(null);
            } else { //否则设置部门文本框为选中员工所在的部门
                TbRecord record = (TbRecord) dao.queryRecordByNum
(selectedItem.substring(0, 6));
                inDeptTextField.setText(record.getTbDutyInfo().
getTbDept().getName());
            }
        }
    }
});
```

### 关键代码解析

❶ itemStateChanged(): 当下拉列表框的选中项发生改变时将触发该方法。  
❷ getStateChange(): 该方法返回一个 int 型值。当返回值等于静态常量 ItemEvent.DESELECTED 时,表示此次事件是由取消原选中项触发的;当返回值等于静态常量 ItemEvent.SELECTED 时,表示此次事件是由选中当前项触发的。

❸ getItem(): 通过该方法可以获得触发此次事件的选项的内容。

如果用户先选中考勤员工所在的部门,例如,选中“经理办公室”,再单击“考勤员工”下拉列表框,在下拉列表框中将显示选中部门的所有员工,如图 17.25 所示。



524







图 17.25 选中部门后再单击“考勤员工”下拉列表框

这是因为在捕获按钮事件弹出部门选取对话框时,根据用户选取的部门对“考勤员工”下拉列表框进行了处理,具体代码如下:

```
final JButton inDeptTreeButton = new JButton(); //创建按钮对象
inDeptTreeButton.addActionListener(new ActionListener() { //捕获按钮事件
    public void actionPerformed(ActionEvent e) {
        //创建部门选取对话框
        DeptTreeDialog deptTree = new DeptTreeDialog(inDeptTextField);
        deptTree.setBounds(375, 317, 101, 175); //设置部门选取对话框的位置
        deptTree.setVisible(true); //弹出部门选取对话框
        //检索选中的部门对象
        TbDept dept = (TbDept) dao.queryDeptByName(inDeptTextField.getText());
        personnalComboBox.removeAllItems(); //清空“考勤员工”下拉列表框中的所有选项
        personnalComboBox.addItem("请选择"); //添加提示项
        //通过 Hibernate 的一对多关联获得与该部门关联的职务信息对象
        Iterator dutyInfoIt = dept.getTbDutyInfos().iterator();
        while (dutyInfoIt.hasNext()) { //遍历职务信息对象
            //获得职务信息对象
            TbDutyInfo dutyInfo = (TbDutyInfo) dutyInfoIt.next();
            //通过 Hibernate 的一对一关联获得与职务信息对象关联的档案信息对象
            TbRecord tbRecord = dutyInfo.getTbRecord();
            personnalComboBox.addItem(tbRecord.getRecordNumber() + " " +
                tbRecord.getName());
        }
    }
});
inDeptTreeButton.setText("...");
```

### 3. 通过 Java 反射验证数据是否为空

在添加培训记录时,所有的培训信息均不允许为空,并且都是通过文本框接收用户输入信息的,在这种情况下可以通过 Java 反射验证数据是否为空,当为空时弹出提示信息,并令为空的文本框获得焦点。具体代码如下:

```
//通过 Java 反射机制获得类中的所有属性
① Field[] fields = BringUpOperatePanel.class.getDeclaredFields();
for (int i = 0; i < fields.length; i++) { //遍历属性数组
    Field field = fields[i]; //获得属性
    //只验证 JTextField 类型的属性
    ② if (field.getType().equals(JTextField.class)) {
        //默认情况下不允许访问私有属性,如果设为 true 则允许访问
        field.setAccessible(true);
        JTextField textField = null;
        try {
            ③ textField = (JTextField) field.get(BringUpOperatePanel.this);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (textField.getText().equals("")) { //查看该属性是否为空
            String infos[] = { "请将培训信息填写完整!", "所有信息均不允许为空!" };
            JOptionPane.showMessageDialog(null, infos, "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
            textField.requestFocus(); //令空的文本框获得焦点
        }
        return;
    }
}
```



525





```
    }  
    }  
}
```

### 🔊 关键代码解析

- ❶ `getDeclaredFields()`: 该方法返回一个 `Field` 型数组, 在数组中包含调用类的所有属性, 包括公共、保护、默认 (包) 访问和私有字段, 但不包括继承的字段。
- ❷ `getType()`: 该方法返回一个 `Class` 对象, 它标识了此属性的声明类型。
- ❸ `BringUpOperatePanel.this`: 代表本类。

## 17.7.4 单元测试

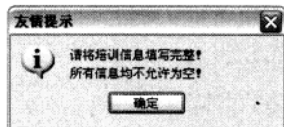


图 17.26 提示培训信息不允许为空的对话框

在测试添加培训记录时, 在不填写任何信息的情况下直接单击“保存”按钮, 并没有弹出如图 17.26 所示的提示培训信息不允许为空的对话框。

查看控制台, 发现抛出了如图 17.27 所示的异常信息, 异常信息中提示不能访问 `BringUpOperatePanel` 类中 `private` 类型的属性, 并且提示是由 `BringUpOperatePanel` 类的第 170 行抛出的, 第 170 行的相关代码如图 17.28 所示。

```
java.lang.IllegalAccessException: Class com.mwq.frame.personnel.BringUpOperatePanel$4  
can not access a member of class com.mwq.frame.personnel.BringUpOperatePanel with modifiers "private"  
at sun.reflect.Reflection.ensureMemberAccess(Unknown Source)  
at java.lang.reflect.Field.doSecurityCheck(Unknown Source)  
at java.lang.reflect.Field.getFieldAccessor(Unknown Source)  
at java.lang.reflect.Field.get(Unknown Source)  
at com.mwq.frame.personnel.BringUpOperatePanel$4.actionPerformed(BringUpOperatePanel.java:170)
```

图 17.27 在控制台抛出的异常信息

```
167 JTextField textField = null;  
168 try {  
169     textField = (JTextField) field  
170     .get(BringUpOperatePanel.this); // 获得本类中的对应属性  
171 } catch (Exception e) {  
172     e.printStackTrace();  
173 }
```

图 17.28 提示发生错误的代码

对照提示信息分析代码, 发现这是因为所有 `JTextField` 类型的属性均声明为私有 (`private`), 而 Java 反射机制默认不允许访问私有属性。在这种情况下可以通过 `java.lang.reflect.Field` 类的 `setAccessible(boolean accessible)` 方法设置私有属性可以访问, 即设置为 `true`, 默认为 `false`。具体代码如下:

```
field.setAccessible(true); // 默认情况下不允许访问私有属性, 如果设置为 true 则允许访问  
JTextField textField = null;  
try {  
    // 获得本类中的对应属性  
    textField = (JTextField) field.get(BringUpOperatePanel.this);  
} catch (Exception e) {
```







```
e.printStackTrace();
}
```

## 17.8 待遇管理模块设计

待遇管理功能是企业人事管理系统的主要功能之一,该功能需要建立在人事管理功能的基础上,例如,人事管理中的考勤管理和奖惩管理在待遇管理中将用到。

### 17.8.1 待遇管理模块功能概述

待遇管理模块包含账套管理、人员设置和统计报表 3 个子模块。账套管理模块用来建立和维护账套信息,包括建立、修改和删除账套,以及为账套添加项目和修改金额,或者从账套中删除项目。首先需要建立一个账套,“新建账套”对话框如图 17.29 所示,其中,账套说明用来详细介绍该账套的适用范围。然后为新建的账套添加项目,“添加项目”对话框如图 17.30 所示,选中要添加的项目后单击“添加”按钮。

最后修改新添加项目的金额,“修改金额”对话框如图 17.31 所示,输入项目金额后单击“确定”按钮。

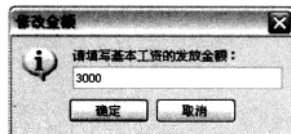
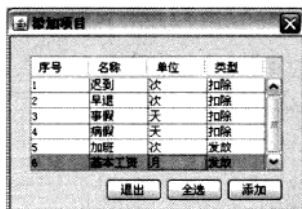
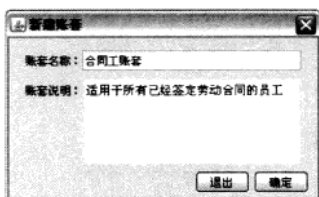


图 17.29 “新建账套”对话框 图 17.30 “添加项目”对话框 图 17.31 “修改金额”对话框

人员设置模块用来设置每个账套具体适合的人员,在这里将用到通过部门树选取员工的对话框。

统计报表模块用来生成员工待遇统计报表,可以生成月、季、半年和年报表,如图 17.32 所示。当生成月报表时,可以选择统计的年和月份;当生成季报表时,可以选择统计的年和季度;当生成半年报时,可以选择统计的年及上半年或下半年;当生成年报时,则只可以选择统计的年。



图 17.32 生成统计报表的种类

### 17.8.2 待遇管理模块技术分析

在实现修改账套项目金额的功能时,通常情况下是通过 JDialog 对话框实现的,但是





因为只需要接收一条修改金额的信息，所以在这里也可以通过 JOptionPane 提示框实现，这样在实现功能的前提下可以少创建一个类，提高了代码的可读性。在开发应用程序时，充分使用提示对话框也是一个不错的选择，既可以帮助用户使用系统，又可以保证系统的安全运行。

### 17.8.3 待遇管理模块实现过程

■ 待遇管理使用的主要数据表：tb\_reckoning、tb\_reckoning\_info、tb\_reckoning\_list。

(1) 在开发账套管理模块时，首先需要建立一个账套，通过弹出对话框获得账套名称和账套说明，将新建的账套添加到左侧的账套表格中，并设置为选中行，还要同步刷新右侧的账套项目表格。具体代码如下：

```
final JButton addSetButton = new JButton();
addSetButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (needSaveRow == -1) { //没有需要保存的账套
            CreateCriterionSetDialog createCriterionSet = new CreateCriterionSetDialog();
            createCriterionSet.setBounds((width - 350) / 2, (height - 250) / 2, 350, 250);
            //弹出“新建账套”对话框，接收账套名称和账套说明
            createCriterionSet.setVisible(true);
            if (createCriterionSet.isSubmit()) { //单击“确定”按钮
                String name = createCriterionSet.getNameTextField().getText();
                //获得账套名称
                String explain = createCriterionSet.getExplainTextArea().
                    getText(); //获得账套说明
                //将新建账套设置为需要保存的账套
                needSaveRow = leftTableValueV.size();
                //创建代表账套表格行的向量对象
                Vector<String> newCriterionSetV = new Vector<String>();
                newCriterionSetV.add(needSaveRow + 1 + ""); //添加账套序号
                newCriterionSetV.add(name); //添加账套名称
                //将向量对象添加到左侧的账套表格中
                leftTableModel.addRow(newCriterionSetV);
                leftTable.setRowSelectionInterval(needSaveRow, needSaveRow);
                //设置新建账套为选中行
                textArea.setText(explain); //设置账套说明
                TbReckoning reckoning = new TbReckoning(); //创建账套对象
                reckoning.setName(name); //设置账套名称
                reckoning.setExplain(explain); //设置账套说明
                reckoningV.add(reckoning); //将账套对象添加到向量中
                refreshItemAllRowValueV(needSaveRow); //同步刷新右侧的账套项目表格
            }
        } else { //有需要保存的账套，弹出提示保存对话框
            JOptionPane.showMessageDialog(null, "请先保存账套: " + leftTable.
                getValueAt(needSaveRow, 1),
                "友情提示", JOptionPane.INFORMATION_MESSAGE);
        }
    }
});
addSetButton.setText("新建帐套");
```

(2) 为新建的账套添加项目，通过弹出对话框获得用户添加的项目，因为需要通过弹出对话框中的表格对象获得选中项目的信息，所以在完成添加项目之前不能销毁添加项目对话框对象，而是将其设置为不可见，添加完成后才能销毁，并且需要判断新添加项目在账套中是否已经存在。具体代码如下：

```
public void addItem(int leftSelectedRow) {
```







```

AddAccountItemDialog addAccountItemDialog = new AddAccountItemDialog();
addAccountItemDialog.setBounds((width - 500) / 2, (height - 375) / 2, 500, 375);
addAccountItemDialog.setVisible(true); //弹出“添加项目”对话框
JTable itemTable = addAccountItemDialog.getTable(); //获得对话框中的表格对象
int[] selectedRows = itemTable.getSelectedRows(); //获得选中行的索引
if (selectedRows.length > 0) { //有新添加的项目
    needSaveRow = leftSelectedRow; //设置当前账套为需要保存的账套
    //将选中行设置为新添加项目的第一行
    int defaultSelectedRow = rightTable.getRowCount();
    //获得选中账套的对象
    TbReckoning reckoning = reckoningV.get(leftSelectedRow);
    for (int i = 0; i < selectedRows.length; i++) { //通过循环向账套中添加项目
        //获得项目名称
        String name = itemTable.getValueAt(selectedRows[i], 1).toString();
        //获得项目单位
        String unit = itemTable.getValueAt(selectedRows[i], 2).toString();
        Iterator<TbReckoningInfo> reckoningInfoIt = reckoning
            .getTbReckoningInfos().iterator(); //遍历账套中的现有项目
        boolean had = false; //默认在现有项目中不包含新添加的项目
        while (reckoningInfoIt.hasNext()) { //通过循环查找是否存在
            TbAccountItem accountItem = reckoningInfoIt.next().
                getTbAccountItem(); //获得已有的项目对象
            if (accountItem.getName().equals(name) && accountItem.getUnit().
                equals(unit)) {
                had = true; //存在
                break; //跳出循环
            }
        }
        if (!had) { //如果没有则添加
            //创建账套信息对象
            TbReckoningInfo reckoningInfo = new TbReckoningInfo();
            //获得账套项目对象
            TbAccountItem accountItem = (TbAccountItem) dao
                .queryAccountItemByNameUnit(name, unit);
            //建立从账套项目到账套信息的关联
            accountItem.getTbReckoningInfos().add(reckoningInfo);
            //建立从账套信息到账套项目的关联
            reckoningInfo.setTbAccountItem(accountItem);
            reckoningInfo.setMoney(0); //设置项目金额为 0
            //建立从账套信息到账套的关联
            reckoningInfo.setTbReckoning(reckoning);
            //建立从账套到账套信息的关联
            reckoning.getTbReckoningInfos().add(reckoningInfo);
        }
    }
    refreshItemAllRowValueV(leftSelectedRow); //同步刷新右侧的账套项目表格
    rightTable.setRowSelectionInterval(defaultSelectedRow,
        defaultSelectedRow); //设置选中行
    addAccountItemDialog.dispose(); //销毁“添加项目”对话框
}
}

```

(3) 为添加的项目修改金额, 如果未添加金额, 是不允许保存的, 因为默认项目金额为 0, 这是没有意义的。这里通过 JOptionPane 提示框获得修改后的金额, 之后还要判断用户输入的金额是否符合要求, 首要条件是数字, 还要求必须为 1~999999 之间的整数。具体代码如下:

```

public void updateItemMoney(int leftSelectedRow, int rightSelectedRow) {
    String money = null;
    done: while (true) {
        ❶ money = JOptionPane.showInputDialog(null, "请填写"
            + rightTable.getValueAt(rightSelectedRow, 1) + "的"
            + rightTable.getValueAt(rightSelectedRow, 3) + "金额: ",
            "修改金额", JOptionPane.INFORMATION_MESSAGE);
        if (money == null) { //用户单击“取消”按钮

```







```
        break done; //取消修改
    } else { //用户单击“确定”按钮
        if (money.equals("")) { //未输入金额，弹出提示对话框
            JOptionPane.showMessageDialog(null, "请输入金额!", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
        } else { //输入了金额
            //金额必须在 1~999999 之间
            Pattern pattern = Pattern.compile("[1-9][0-9]{0,5}");
            //通过正则表达式判断是否符合要求
            Matcher matcher = pattern.matcher(money);
            if (matcher.matches()) { //符合要求
                needSaveRow = leftSelectedRow; //设置当前账套为需要保存的账套
                //修改项目金额
                rightTable.setValueAt(money, rightSelectedRow, 4);
                //默认存在下一行
                int nextSelectedRow = rightSelectedRow + 1;
                //存在下一行
                if (nextSelectedRow < rightTable.getRowCount()) {
                    rightTable.setRowSelectionInterval(nextSelectedRow,
                        nextSelectedRow);
                }
                String name = rightTable.getValueAt(rightSelectedRow, 1).
                    toString(); //获得项目名称
                String unit = rightTable.getValueAt(rightSelectedRow, 2).
                    toString(); //获得项目单位
                //获得选中账套的对象
                TbReckoning reckoning = reckoningV.get(leftSelectedRow);
                Iterator reckoningInfoIt = reckoning.
                    getTbReckoningInfos().iterator(); //遍历项目
                while (reckoningInfoIt.hasNext()) { //通过循环查找选中的项目
                    TbReckoningInfo reckoningInfo = (TbReckoningInfo)
                        reckoningInfoIt.next();
                    TbAccountItem accountItem = reckoningInfo.
                        getTbAccountItem();
                    if (accountItem.getName().equals(name) && accountItem.
                        getUnit().equals(unit)) {
                        //修改金额
                        reckoningInfo.setMoney(new Integer(money));
                        break; //跳出循环
                    }
                }
                break done; //修改完成
            } else { //不符合要求，弹出提示对话框
                String infos[] = { "金额输入错误，请重新输入!", "金额必须
                    为 0~999999 之间的整数!" };
                JOptionPane.showMessageDialog(null, infos, "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }
}
```

### 关键代码解析



❶ showInputDialog(): 该静态方法用来弹出提示某些输入的提示框，可以接受用户的输入信息，也可以提供几个备选项供用户选择。

❷ showMessageDialog(): 该静态方法用来弹出提示某些消息的提示框，消息的类型可以为错误 (ERROR\_MESSAGE)、消息 (INFORMATION\_MESSAGE)、警告 (WARNING\_MESSAGE)、问题 (QUESTION\_MESSAGE) 或普通 (PLAIN\_MESSAGE)。

❸ infos[]: 如果需要将消息分多行显示，可以将消息放到数组中，每一个数组索引代表一行。





(4) 开发统计报表, 首先判断报表类型, 然后根据报表类型组织报表的起止时间。下面是生成季度报表的代码:

```
String quarter = quarterComboBox.getSelectedItem().toString(); //获得季度报表
if (quarter.equals("第一")) {
    reportForms(year + "-1-1", year + "-3-31"); //生成报表
} else if (quarter.equals("第二")) {
    reportForms(year + "-4-1", year + "-6-30"); //生成报表
} else if (quarter.equals("第三")) {
    reportForms(year + "-7-1", year + "-9-30"); //生成报表
} else {
    reportForms(year + "-10-1", year + "-12-31"); //生成报表
}
```

下面的代码负责在生成报表时向表格中添加员工的关键信息, 初始实发金额为 0 元。

```
TbRecord record = (TbRecord) recordIt.next(); //获得档案对象
Vector recordV = new Vector(); //创建与档案对象对应的向量
recordV.add(num++); //添加序号
recordV.add(record.getRecordNumber()); //添加档案编号
recordV.add(record.getName()); //添加姓名
recordV.add(record.getSex()); //添加性别
TbDutyInfo dutyInfo = record.getTbDutyInfo();
recordV.add(dutyInfo.getTbDept().getName()); //添加部门
recordV.add(dutyInfo.getTbDuty().getName()); //添加职务
int salary = 0; //初始实发金额为 0
```

下面的代码负责在生成报表时计算员工的奖惩金额, 通过 Hibernate 的关联得到的是员工的所有奖惩, 需要通过集合过滤功能进行过滤, 检索符合条件的奖惩信息。具体代码如下:

```
Set rewAndPuns = record.getTbRewardsAndPunishmentsForRecordId();
String types[] = new String[] { "奖励", "惩罚" };
for (int i = 0; i < types.length; i++) {
    String filterHql = "where this.type='" + types[i] + "' and ( ( startDate between '" + reportStartDateStr + "' and '" + reprotEndDateStr + "' or endDate between '" + reportStartDateStr + "' and '" + reprotEndDateStr + "' ) or ( ' " + reportStartDateStr + "' between startDate and endDate and ' " + reprotEndDateStr + "' between startDate and endDate ) )"; //组织用来过滤集合的 HQL 语句
    List list = dao.filterSet(rewAndPuns, filterHql); //过滤奖惩记录
    if (list.size() > 0) {
        column += 1; //存在奖惩
        int money = 0; //列索引加 1
        for (Iterator it = list.iterator(); it.hasNext(); ) { //初始奖惩金额为 0
            TbRewardsAndPunishment rewAndPun = (TbRewardsAndPunishment) it.next();
            money += rewAndPun.getMoney(); //累加奖惩金额
        }
        recordV.add(money); //添加奖惩金额
        if (i == 0) //奖励
            salary += money; //计算实发金额
        else //惩罚
            salary -= money; //计算实发金额
    } else {
        recordV.add("-"); //没有奖励或惩罚
    }
}
```





## 17.9 系统维护模块设计

系统维护模块用来维护系统的基本信息，例如，企业架构信息和常用的职务种类、用工形式等信息，另外，在该模块还提供了对系统进行初始化的功能。

### 17.9.1 系统维护模块功能概述

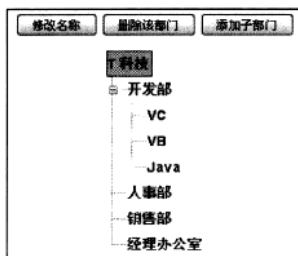


图 17.33 企业架构主界面

系统维护模块包含企业架构、基本资料和初始化系统 3 个子模块。

企业架构模块用来维护企业的组织结构信息，包括修改公司及部门的名称，添加或删除部门。企业架构主界面如图 17.33 所示。

可以选中公司或部门后单击“修改名称”按钮，修改公司或部门的名称。如果修改的是公司名称，将弹出如图 17.34 所示的界面；如果修改的是部门名称，将弹出如图 17.35 所示的界面。

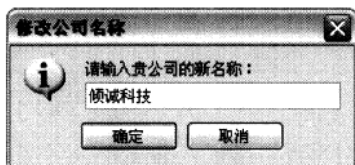


图 17.34 修改公司名称

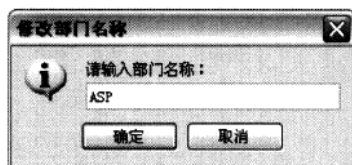


图 17.35 修改部门名称

也可以为公司或部门添加子部门。如果是在公司下添加子部门，则选中公司节点；如果是在公司所属部门下添加子部门，则选中所属部门，然后单击“添加子部门”按钮，将弹出如图 17.36 所示的界面，在二级部门（例如图 17.33 中的 Java）下不能再包含子部门，如果试图在该级部门下建立子部门，将弹出如图 17.37 所示的提示框。

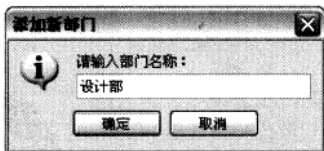


图 17.36 为公司或一级部门添加子部门

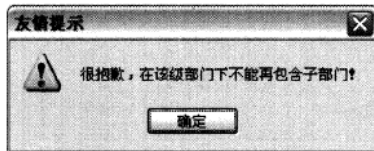


图 17.37 不允许在二级部门下添加子部门

还可以选中部门节点后单击“删除该部门”按钮删除选中的部门，在删除之前将弹出如图 17.38 所示的提示框。需要注意的是，公司节点不允许删除，如果试图删除公司，将弹出如图 17.39 所示的提示框。





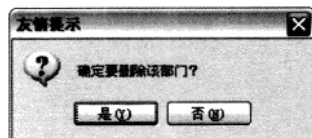


图 17.38 询问是否删除部门

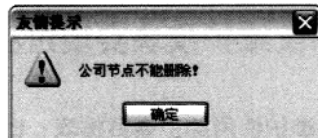


图 17.39 提示公司不允许删除

基本资料模块用来维护系统中的基本信息,例如,职务种类、账套项目等,如图 17.40 所示。

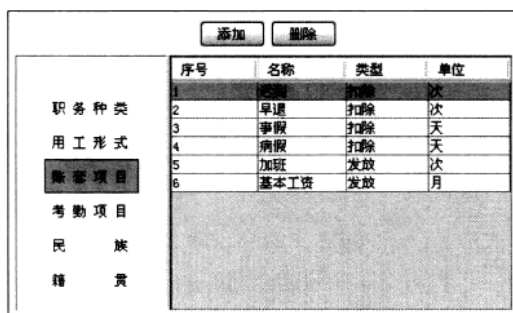


图 17.40 维护基本资料界面

初始化系统模块用来对系统进行初始化,用户在使用系统之前,或者是想清空系统中的数据,可以通过该功能实现。不过,在对系统进行初始化之前,一定要弹出一个如图 17.41 所示的提示框询问是否初始化,这样会增加系统的安全性,因为可能是用户不小心点到的。初始化完成后,将弹出如图 17.42 所示的提示框,该提示框的内容为本系统的使用步骤。

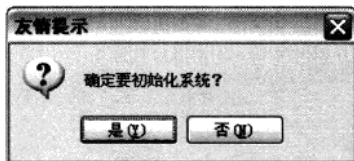


图 17.41 询问是否初始化系统

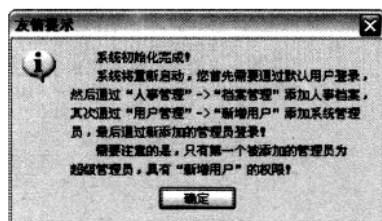


图 17.42 初始化完成后弹出的提示框

## 17.9.2 系统维护模块技术分析

维护企业架构是该模块的技术难点。因为企业架构是一个树状结构,所以需要通过 Swing 中的 JTree 组件完成。对企业架构的维护主要包括修改公司或部门的名称、设立新部门或取消现有部门,这对于 JTree 组件来说,就是修改节点名称、添加新节点或删除现有节点。为了实现对上述树节点的操作,还需要一些其他的操作 JTree 组件的知识,例如,如何获得选中树节点的路径,如何获得选中树节点的对象,如何展开指定的树节点等。



### 17.9.3 系统维护模块实现过程

■ 系统维护使用的主要数据表：tb\_dept。

维护企业架构是该模块的技术难点，所以在这里只介绍企业架构模块的实现过程。首先实现修改名称的功能，分为修改公司名称和修改部门名称。在修改公司名称之前弹出提示询问是否真的修改，在修改部门名称时则不询问，直接弹出消息框供用户输入新名称。修改时需要分两步实现，第一步是修改系统界面的企业架构树，第二步是修改持久化对象，并持久化到数据库中。具体代码如下：

```
final JButton updateButton = new JButton();
updateButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ❶ TreePath selectionPath = tree.getSelectionPath();
        TbDept selected = null;
        String newName = null;
        ❷ if (selectionPath.getPathCount() == 1) { //修改公司名称
            int i = JOptionPane.showConfirmDialog(null, "确定要修改贵公司的名称?",
                "友情提示", JOptionPane.YES_NO_OPTION); //弹出提示框
            if (i == 0) { //修改（单击“是”按钮）
                String infos[] = { "请输入贵公司的新名称:", "修改公司名称", "请输入贵
公司的新名称!" };
                newName = getName(infos); //获得修改后的名称
                if (newName != null) {
                    selected = company; //修改的为公司名称
                }
            } else { //修改部门名称
                String infos[] = { "请输入部门的新名称:", "修改部门名称", "请输入部门的新
名称!" };
                newName = getName(infos); //获得修改后的名称
                if (newName != null) {
                    selected = company; //选中部门的所属部门
                }
            }
            //选中部门节点的路径对象
            ❸ Object[] paths = selectionPath.getPath();
            for (int i = 1; i < paths.length; i++) { //遍历选中节点路径
                Iterator deptIt = selected.getTbDepts().iterator();
                //通过循环查找选中节点路径对应的部门
                found: while (deptIt.hasNext()) {
                    TbDept dept = (TbDept) deptIt.next();
                    if (dept.getName().equals(paths[i].toString())) {
                        selected = dept; //找到选中节点路径对应的部门
                        break found; //跳出到指定位置
                    }
                }
            }
        }
        if (selected != null) {
            ❹ DefaultMutableTreeNode treeNode = (DefaultMutableTreeNode)
selectionPath
                .getLastPathComponent(); //获得选中节点对象
            ❺ treeNode.setUserObject(newName); //修改节点名称
            ❻ treeNode.reload(); //刷新树结构
            ❼ tree.setSelectionPath(selectionPath); //设置节点为选中状态
            selected.setName(newName); //修改部门对象
            dao.updateObject(company); //将修改持久化到数据库
            HibernateSessionFactory.closeSession(); //关闭数据库连接
        }
    }
});
updateButton.setText("修改名称");
```





### 关键代码解析

❶ `getSelectionPath()`: 通过该方法可以获得选中节点的路径对象, 通过该对象可以获得选中节点的相关信息, 例如, 选中节点对象、选中节点级别等。

❷ `getPathCount()`: 通过该方法可以获得选中节点的级别, 当返回值为 1 时, 代表选中的为树的根节点, 为 2 时则代表选中的是树的直属子节点。

❸ `getPath()`: 通过该方法可以获得选中节点路径包含的节点对象, 返回值为 `Object` 型的数组。例如, `A` 为树的根节点, `a1` 和 `a2` 为 `A` 的子节点, 如果选中 `a2`, 则返回的为 `Object nodes[]={A, a2}`。

❹ `getLastPathComponent()`: 通过该方法可以获得选中节点的对象。例如, `A` 为树的根节点, `a1` 和 `a2` 为 `A` 的子节点, 如果选中 `a2`, 则返回的为 `a2`。

❺ `setUserObject(Object title)`: 该方法用来设置节点的标题。

❻ `reload()`: 该方法用来刷新已经被修改的树模型对象。

❼ `setSelectionPath(TreePath path)`: 该方法用来设置选中指定路径的节点。

实现添加部门的功能, 只允许在公司及其直属部门下添加部门, 在获得用户输入的要添加部门的名称之后, 需要判断在其所属部门中是否存在该部门, 如果存在则弹出提示, 否则添加到系统界面的企业架构树中, 并持久化到数据库。具体代码如下:

```
final JButton addButton = new JButton();
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        TreePath selectionPath = tree.getSelectionPath();
        int pathCount = selectionPath.getPathCount(); //获得选中节点的级别
        had: if (pathCount == 3) { //选中的为 3 级节点
            JOptionPane.showMessageDialog(null, "很抱歉, 在该级部门下不能再包含子部门!",
                "友情提示", JOptionPane.WARNING_MESSAGE);
        } else { //选中的为 1 级或 2 级节点
            String infos[] = { "请输入部门名称: ", "添加新部门", "请输入部门名称!" };
            String newName = getName(infos); //获得新部门的名称
            if (newName != null) { //创建新部门
                DefaultMutableTreeNode parentNode = (DefaultMutableTreeNode)
                    selectionPath
                        .getLastPathComponent(); //获得选中部门节点对象
                //获得该部门包含子部门的个数
                int childCount = parentNode.getChildCount();
                for (int i = 0; i < childCount; i++) { //查看新创建的部门是否已经存在
                    TreeNode childNode = parentNode.getChildAt(i);
                    if (childNode.toString().equals(newName)) {
                        JOptionPane.showMessageDialog(null, "该部门已经存在!",
                            "友情提示", JOptionPane.WARNING_MESSAGE);
                        break had; //已经存在, 跳出到指定位置
                    }
                }
                DefaultMutableTreeNode childNode = new
                    DefaultMutableTreeNode(newName); //创建部门节点
                treeModel.insertNodeInto(childNode, parentNode, childCount);
                //插入新部门到选中部门的最后
                tree.expandPath(selectionPath); //展开指定路径中的尾节点
                TbDept selected = company; //默认选中的为 1 级节点
                if (pathCount == 2) { //选中的为 2 级节点
                    //获得选中节点的名称
                    String selectedName = selectionPath.getPath()[1].toString();
                    //创建公司直属部门的迭代器对象
                    Iterator deptIt = company.getTbDepts().iterator();
                    found: while (deptIt.hasNext()) { //遍历公司的直属部门
                        TbDept dept = (TbDept) deptIt.next();
                        //查找与选中节点对应的部门
```





```
        if (dept.getName().equals(selectedName)) {
            selected = dept;          //设置为选中部门
            break finded;              //跳出循环
        }
    }

    TbDept sonDept = new TbDept();    //创建新部门对象
    sonDept.setName(newName);         //设置部门名称
    sonDept.setTbDept(selected);      //建立从新部门到所属部门的关联
    selected.getTbDepts().add(sonDept); //建立从所属部门到新部门的关联
    dao.updateObject(company);         //将新部门持久化到数据库
    HibernateSessionFactory.closeSession(); //关闭数据库连接
}

});
addButton.setText("添加子部门");
```

### 🔊 关键代码解析

- ❶ getChildCount(): 通过该方法可以获得包含子节点的个数。
- ❷ getChildAt(int childIndex): 该方法用来获得指定索引位置的子节点, 索引从 0 开始。
- ❸ insertNodeInto(MutableTreeNode newChild, MutableTreeNode parent, int index): 该方法用来将新创建的子节点 newChild 插入到父节点 parent 中索引为 index 的位置。
- ❹ expandPath(TreePath path): 该方法用来展开指定路径中的尾节点, 前提条件是指定路径的尾节点不能是叶子节点。

最后实现删除现有部门的功能。公司节点不允许删除, 如果试图删除公司节点, 将弹出不允许删除的提示, 在删除部门之前也将弹出提示框询问是否确定删除, 如果确定删除, 则从系统界面的企业架构树中删除选中部门, 并持久化到数据库中。具体代码如下:

```
final JButton delButton = new JButton();
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        TreePath selectionPath = tree.getSelectionPath();
        int pathCount = selectionPath.getPathCount(); //获得选中节点的级别
        if (pathCount == 1) {
            //选中的为 1 级节点, 即公司节点
            JOptionPane.showMessageDialog(null, "公司节点不能删除!", "友情提示",
                JOptionPane.WARNING_MESSAGE);
        } else {
            //选中的为 2 级或 3 级节点, 即部门节点
            DefaultMutableTreeNode treeNode = (DefaultMutableTreeNode)
                selectionPath
                    .getLastPathComponent(); //获得选中部门节点对象
            int i = JOptionPane.showConfirmDialog(null, "确定要删除该部门: "
                + treeNode, "友情提示", JOptionPane.YES_NO_OPTION);
            if (i == 0) {
                //删除
                treeModel.removeNodeFromParent(treeNode); //删除选中节点
                tree.setSelectionRow(0); //选中根(公司)节点
                TbDept selected = company; //选中部门的所属部门
                Object[] paths = selectionPath.getPath(); //选中部门节点的路径对象
                int lastIndex = paths.length - 1; //获得最大索引
                for (int j = 1; j <= lastIndex; j++) { //遍历选中节点路径
                    Iterator deptIt = selected.getTbDepts().iterator();
                    //通过循环查找选中节点路径对应的部门
                    finded: while (deptIt.hasNext()) {
                        TbDept dept = (TbDept) deptIt.next();
                        if (dept.getName().equals(paths[j].toString())) {
                            if (j == lastIndex) //如果为选中节点
                                //删除选中部门
                                selected.getTbDepts().remove(dept);
                            else //如果为所属节点
                                selected = dept;
                        }
                    }
                }
            }
        }
    }
});
```





```

        break finded; //跳出到指定位置
    }
}
dao.updateObject(company); //同步删除数据库
HibernateSessionFactory.closeSession(); //关闭数据库连接
}
});
delButton.setText("删除该部门");

```

### 关键代码解析

- ❶ removeNodeFromParent(MutableTreeNode node): 该方法用来从树模型中移除指定节点。
- ❷ setSelectionRow(int row): 该方法用来设置选中的行。树的根节点为第 0 行。例如, A 为树的根节点, a1 和 a2 为 A 的子节点, s 为 a1 的子节点, 如果 a1 节点处于合并状态, 则 a2 为第 2 行; 如果 a1 节点处于展开状态, 则 a2 为第 3 行。

## 17.9.4 单元测试

开发完成企业架构模块后, 需要对该模块进行整体测试, 这是软件开发过程中的最后一步, 也是最关键的一步。

首先测试添加新部门, 假设公司需要设立一个后勤部, 单击“添加子部门”按钮, 在弹出的对话框中输入“后勤部”后单击“确定”按钮, “后勤部”节点即成功添加到树中。

下面测试修改部门名称, 将“后勤部”修改为“后勤服务部”。单击“修改名称”按钮, 在弹出的对话框中输入“后勤服务部”后单击“确定”按钮, 发现在树中仍为“后勤部”, 查看控制台, 并没有抛出任何异常信息, 重新运行程序, 发现“后勤部”已经变为了“后勤服务部”, 这说明已经修改成功, 可是为什么在修改结束后树中并没有改变呢? 继续测试删除现有部门, 选中“后勤服务部”后单击“删除该部门”按钮, 在弹出的提示框中单击“是”按钮, 发现删除功能也实现了, 在树中已经没有“后勤服务部”节点了。

修改名称后在树结构中并没有显示新名称, 但是重新运行系统后却显示了修改后的名称, 说明已经修改成功了, 只是未刷新页面的缘故。

在添加和删除节点时, 都是通过 DefaultTreeModel 类的对象 treeModel 实现的。关键代码如下:

```

treeModel.removeNodeFromParent(treeNode);
treeModel.insertNodeInto(childNode, parentNode, childCount);

```

修改节点名称则是通过 DefaultMutableTreeNode 类的对象 treeNode 实现的。关键代码如下:

```

treeNode.setUserObject(newName);

```

在通过 DefaultTreeModel 类的对象修改树节点时, 会同步刷新系统界面, 而通过 DefaultMutableTreeNode 类的对象修改节点时, 则不会同步刷新系统界面, 必须通过 DefaultTreeModel 类的对象进行手动刷新, 这样修改后在系统界面就会显示为修改后的内容了。关键代码如下:

```

treeModel.reload();

```





## 17.10 开发问题解析

由 `JTable` 类实现的表格是可编辑的，并且表格列允许重新排列。而本系统使用的所有表格都不需要表格的可编辑功能，并且多数表格拥有的列数都很少，也不需要表格列的重新排列功能。基于这种情况，本系统通过继承 `JTable` 类并重写部分方法，实现了自己的表格组件类 `MTable`，通过 `MTable` 类创建的表格是不可编辑的，并且表格列不允许重新排列。

`JTable` 类的 `isCellEditable()` 方法返回一个 `boolean` 型值，当返回 `true` 时，说明表格是可编辑的，所以只需要重写该方法，直接返回 `false`，这样表格就不可编辑了；通过 `JTable` 类的 `getTableHeader()` 方法可以获得 `JTableHeader` 类的对象，通过该对象可以设置表头的相关信息，包括设置表格列是否允许重新排列，设置的方法是 `setReorderingAllowed(boolean isAllowed)`，当该方法的入口参数为 `true` 时，表格列允许重新排列，为 `false` 时则不允许重新排列。`MTable` 类的完整代码如下：

```
public class MTable extends JTable {
    public MTable(DefaultTableModel tableModel) {
        super(tableModel); // 继承超类的构造方法
    }
    // 重写该方法，直接返回 false，令表格不可编辑
    public boolean isCellEditable(int row, int column) {
        return false;
    }
    public JTableHeader getTableHeader() {
        // 通过超类方法获得 JTableHeader 类的对象
        JTableHeader tableHeader = super.getTableHeader();
        tableHeader.setReorderingAllowed(false); // 设置表格列不允许重新排列
        return tableHeader;
    }
}
```

## 17.11 Hibernate 关联关系的建立方法

在本系统中有多处用到了 Hibernate 的一对一和一对多关联，通过对 Hibernate 关联关系的使用，可以快速地通过一个对象获得与之关联的对象。下面就介绍一下这两种关联方式的配置方法。

### 17.11.1 建立一对一关联

本系统将档案信息和职务信息分别保存到了两个表中，如图 17.43 所示，与这两个表对应的持久化类为 `TbRecord` 和 `TbDutyInfo`，在这两个持久化类之间就用到了 Hibernate 的一对一关联，因为本系统只允许一个员工担任一个职务。

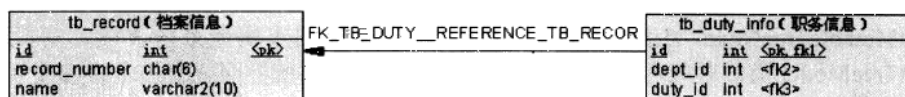


图 17.43 一对一关联模型





(1) 在拥有主键的持久化类 TbRecord 中创建一个关联类 TbDutyInfo 的对象 tbDutyInfo, 以及其对应的 set/get 方法, 具体代码如下:

```
private TbDutyInfo tbDutyInfo;
public TbDutyInfo getTbDutyInfo() {
    return tbDutyInfo;
},
public void setTbDutyInfo(TbDutyInfo tbDutyInfo) {
    this.tbDutyInfo = tbDutyInfo;
}
```

(2) 在持久化类 TbRecord 的映射文件 TbRecord.hbm.xml 中添加如下代码:

```
<one-to-one name="tbPersonalInfo" class="com.mwq.hibernate.mapping.
TbPersonalInfo" cascade="all" />
```

<one-to-one>元素用来映射持久化类之间的一对一关联关系, 其中, name 属性为持久化类中关联类的对象, class 属性为关联类的类型, cascade 属性用来设置对关联对象的操作级别, 当设为 all 时, 表示当保存、修改或删除当前对象时, 将级联保存、修改或删除关联对象。

(3) 在关联类 TbDutyInfo 中创建一个 TbRecord 类的对象 tbRecord 及其对应的 set/get 方法, 并且在相应的映射文件中添加如下代码, 当将<one-to-one>元素的 constrained 属性设置为 true 时, 说明该类的主键将同时作为外键, 参照关联类的主键。

```
<one-to-one name="tbRecord" class="com.mwq.hibernate.mapping.TbRecord"
constrained="true" />
```

既然关联类的主键将同时作为外键, 就要修改关联类的主键的映射代码。修改后的代码如下:

```
<id name="id" type="java.lang.Integer">
    <column name="id" />
    ❶ <generator class="foreign">
    ❷ <param name="property">tbRecord</param>
    </generator>
</id>
```

### 🔊 关键代码解析

❶ foreign: 将 class 属性设置为 foreign 时, 表示该主键将同时作为外键, 所以主键的生成方式将参考外键。

❷ <param>: 这里用来设置外键的参考信息, 参考的为关联对象 tbRecord 的主键的值。

## 17.11.2 建立一对多关联

虽然本系统只允许一个员工担任一个职务, 即一个员工只能属于一个部门, 但是一个部门却可以拥有多个员工, 所以部门和员工之间是一对多的关系, 即在持久化类 TbDept 和 TbDutyInfo 之间存在着 Hibernate 的一对多关联关系, 如图 17.44 所示。下面将探讨 Hibernate 一对多关联关系的建立方法。



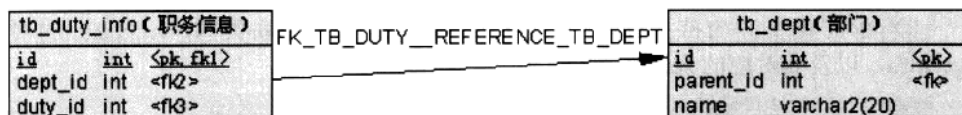


图 17.44 一对多关联模型

(1) 在持久化类 TbDutyInfo 中创建一个关联类 TbDept 的对象 tbDept 及其对应的 set/get 方法, 并且在相应的映射文件中添加如下代码:

```
① <many-to-one name="tbDept" class="com.mwq.hibernate.mapping.TbDept"
  fetch="select" lazy="false">
② <column name="dept_id" not-null="true" />
</many-to-one>
```

#### 关键代码解析

- ① <many-to-one>: 该元素用来映射一对多关联关系中的一方。
- ② <column>: 这里用来设置本类对应表中参考关联类对应表主键的外键列的名称。

(2) 在关联类 TbDept 中创建一个集合类 java.util.Set 的对象 tbDutyInfos, 以及其对应的 set/get 方法, 并且在相应的映射文件中添加如下代码:

```
① <set name="tbDutyInfos" lazy="false">
② <key column="dept_id" />
③ <one-to-many class="com.mwq.hibernate.mapping.TbDutyInfo" />
</set>
```

#### 关键代码解析

① <set>: 该元素用来映射一对多关联关系中的多方, 同时表明该元素的 name 属性值的类型为 java.util.Set; lazy 属性用来设置对关联对象的检索策略, 当设置为 false 时表示立即检索关联对象, 默认为 true, 即只有当访问关联对象时才检索。

② <key>: 这里用来设置关联类对应表中参考本类对应表主键的外键的名称。

③ <one-to-many>: 该元素的 class 属性的值为关联类的名称, 同时也表明 Set 集合中存放对象的类型。

至此, 一个一对多关联就建立完成了。上面建立的是普通的一对多关联, 还有一种特殊的一对多关联, 就是一对多自关联。所谓自关联, 就是外键参考的为本表中的主键。建立方法与普通的一多关联完全相同, 只是对初学者来说有些难于理解。

通常情况下一个企业的组织架构是呈树状的, 即在一个部门当中还可能包含几个下级部门, 本系统就支持这种情况, 所以本系统中用来保存部门信息的表结构如图 17.45 所示。

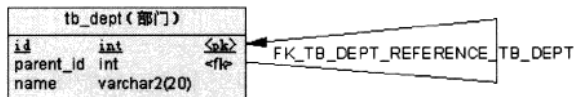


图 17.45 特殊一对多关联模型

针对图 17.45 这种情况建立的一对多关联, 就叫做一对多自关联。在这种情况下, 在与表 tb\_dept 对应的持久化类 TbDept 中既要包含 TbDept 类的对象 tbDept, 用来存放该部门对象所属的上级部门对象; 又要包含集合类 java.util.Set 的对象 tbDepts, 用来存放该部







部门对象包含的下级部门对象。具体代码如下:

```
private TbDept tbDept;  
private Set tbDepts = new HashSet(0);  
public TbDept getTbDept() {  
    return this.tbDept;  
}  
public void setTbDept(TbDept tbDept) {  
    this.tbDept = tbDept;  
}  
public Set getTbDepts() {  
    return this.tbDepts;  
}  
public void setTbDepts(Set tbDepts) {  
    this.tbDepts = tbDepts;  
}
```

同样,在持久化类 TbDept 的映射文件中既要包含<many-to-one>元素,用来映射对象 tbDept;又要包含<set>元素,用来映射对象 tbDepts,具体代码如下:

```
<many-to-one name="tbDept" class="com.mwq.hibernate.mapping.TbDept" fetch="select">  
    <column name="parent_id" not-null="false" />  
</many-to-one>  
<set name="tbDepts" lazy="false" cascade="all-delete-orphan">  
    <key column="parent_id" />  
    <one-to-many class="com.mwq.hibernate.mapping.TbDept" />  
</set>
```